

Up-To Techniques for Weighted Systems (Extended Version)^{*}

Filippo Bonchi¹, Barbara König², and Sebastian Küpper²

¹ ENS Lyon, France

² Universität Duisburg-Essen, Germany

Abstract. We show how up-to techniques for (bi-)similarity can be used in the setting of weighted systems. The problems we consider are language equivalence, language inclusion and the threshold problem (also known as universality problem) for weighted automata. We build a bisimulation relation on the fly and work up-to congruence and up-to similarity. This requires to determine whether a pair of vectors (over a semiring) is in the congruence closure of a given relation of vectors. This problem is considered for rings and l -monoids, for the latter we provide a rewriting algorithm and show its confluence and termination. We then explain how to apply these up-to techniques to weighted automata and provide runtime results.

1 Introduction

Language equivalence of deterministic automata can be checked by means of the bisimulation proof principle. For non-deterministic automata, this principle is sound but not complete: to use bisimulation, one first has to determinize the automaton, via the so-called powerset construction. Since the determinized automaton might be much larger than the original non-deterministic one, several algorithms [24,13,1,8] have been proposed to perform the determinization on the fly and to avoid exploring a huge portion of states. Among these, the algorithm in [8] that exploits *up-to techniques* is particularly relevant for our work.

Up-to techniques have been introduced by Robin Milner in his seminal work on CCS [19] and, since then, they proved useful, if not essential, in numerous proofs about concurrent systems (see [20] for a list of references). According to the standard definition a relation R is a bisimulation whenever two states x, y in R can simulate each other, resulting in a pair x', y' that is still in R . An up-to technique allows to replace the latter R by a larger relation $f(R)$ which contains more pairs and hence allows to cut off bisimulation proofs and work with much smaller relations.

Here we focus on up-to techniques in a quantitative setting: weighted systems, especially weighted automata over arbitrary semirings. Some examples of up-to techniques for weighted systems already appeared in [7] and [21], that study up-to techniques from the abstract perspective of coalgebras.

Although up-to techniques for weighted systems have already received some attentions, their relevance for algorithms to perform behavioural analysis has never been

^{*} Research partially supported by DFG project BEMEGA and ANR-16-CE25-0011 REPAS.

studied properly. This is the main aim of our paper: we give a uniform class of algorithms exploiting up-to techniques to solve the problems of equivalence, inclusion and universality, which, in the weighted setting, asks whether the weight of all words is below some given threshold. In particular we show how to implement these techniques and we perform runtime experiments.

The key ingredient to algorithmically exploit up-to techniques is a procedure to decide, given x, y, R as above, whether x, y belongs to $f(R)$. For a non-deterministic automaton (NFA) with state space S , the algorithm in [8] uses as sub-routine a rewriting system to check whether two sets of states $S, S' \in \mathcal{P}(X)$ – representing states of the determinised automaton – belong to $c(R)$, the congruence closure of R .

For NFA, the congruence closure is taken with respect to the structure of join semi-lattices $(\mathcal{P}(X), \cup, \emptyset)$, carried by the state space of a determinized automaton. For weighted automata, rather than join semi-lattices, we need to consider the congruence closure for *semimodules* (which resemble vector spaces, but are defined over semirings instead of fields). Indeed, an analogon of the powerset construction for weighted automata results in a sort of “determinised automaton” (called in [6] linear weighted automaton) whose states are vectors with values in the underlying semiring.

Our first issue is to find a procedure to check whether two vectors belong to the congruence closure (with respect to semimodules) of a given relation. We face this problem for different semirings, especially rings and l -monoids. For l -monoids we adapt the rewriting procedure for the non-deterministic case [8] and show its confluence and termination, which guarantees a unique normal form as a representative for each equivalence class. Confluence holds in general and termination can be shown for certain semirings, such as the tropical semiring (also known as the $(\min, +)$ -semiring).

Reasoning up-to congruence is sound for language equivalence, but not for inclusion. For the latter, we need the precongruence closure that, in the case of l -monoids, can be checked with a simple modification of the rewriting procedure. Inspired by [1], we further combine this technique with a certain notion of weighted *similarity*, a pre-order that entails language inclusion and can be computed in polynomial time.

We then show how to apply our up-to techniques to language equivalence and inclusion checks for weighted automata. For some interesting semirings, such as the tropical semiring, these problems are known to be undecidable [18]. But based on the inclusion algorithm we can develop an algorithm which solves the universality (also called threshold) problem for the tropical semiring over the natural numbers. This problem is known to be PSPACE-complete and we give detailed runtime results that compare our up-to threshold algorithm with one previously introduced in [3].

2 Preliminaries

In this section we recall all the algebraic structures we intend to work with and, in particular, spaces of vectors over these structures.

A *semiring* is a tuple $\mathbb{S} = (S, +, \cdot, 0, 1)$ where $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, 0 *annihilates* \cdot (i.e., $0 \cdot s_1 = 0 = s_1 \cdot 0$) and \cdot *distributes over* $+$ (i.e., $(s_1 + s_2) \cdot s_3 = s_1 \cdot s_3 + s_2 \cdot s_3$ and $s_3 \cdot (s_1 + s_2) = s_3 \cdot s_1 + s_3 \cdot s_2$). A *ring* is a semiring equipped with inverses for $+$.

Let (L, \sqsubseteq) be a partially ordered set. If for all pairs of elements $\ell_1, \ell_2 \in L$ the infimum $\ell_1 \sqcap \ell_2$ and the supremum $\ell_1 \sqcup \ell_2$ exist (wrt. the order \sqsubseteq), it is a *lattice*. If $(\ell_1 \sqcup \ell_2) \sqcap \ell_3 = (\ell_1 \sqcap \ell_3) \sqcup (\ell_2 \sqcap \ell_3)$ for all $\ell_1, \ell_2, \ell_3 \in L$, it is called *distributive*. It is *complete* if suprema and infima of arbitrary subsets exist. Every complete distributive lattice is a semiring $(L, \sqcup, \sqcap, \perp, \top)$, where \perp, \top are the infimum and supremum of L .

Let (L, \sqsubseteq) be a lattice and $(L, \cdot, 1)$ be a monoid. If \cdot distributes over \sqcup , we call (L, \sqcup, \cdot) an *l-monoid*. Moreover, if L has a \perp -element 0 that annihilates \cdot , we call (L, \sqcup, \cdot) bounded. and it is then a semiring $(L, \sqcup, \cdot, 0, 1)$. It is called *completely distributive* if (L, \sqsubseteq) is complete and multiplication distributes over arbitrary suprema. Observe that every completely distributive *l-monoid* is bounded.³ It is called *integral* if $\top = 1$.

Example 2.1. The tropical semiring is the structure $\mathbb{T} = (\mathbb{R}_0^+ \cup \{\infty\}, \min, +, \infty, 0)$.⁴ \mathbb{T} is a distributive *l-monoid* for the lattice $(\mathbb{R}_0^+ \cup \{\infty\}, \geq)$.

Another example for a distributive *l-monoid* is $\mathbb{M} = ([0, 1], \max, \cdot, 0, 1)$, which is based on the lattice $([0, 1], \leq)$.

The *l-monoid* \mathbb{M} is isomorphic to \mathbb{T} via the isomorphism $\varphi: \mathbb{T} \rightarrow \mathbb{M}, x \mapsto 2^{-x}$.

Hereafter, we will sometimes identify the semiring \mathbb{S} with the underlying set S . For the sake of readability, we will only consider commutative semirings, i.e., semirings where multiplication is commutative.

For a semiring \mathbb{S} and a finite set X , an *\mathbb{S} -vector of dimension X* is a mapping $v: X \rightarrow \mathbb{S}$. The set of all such vectors is denoted by \mathbb{S}^X and is called a *semimodule*.

For notational convenience, we assume that $X = \{1, 2, \dots, |X|\}$ and we write a vector v as a column vector. For X and Y finite sets, an *\mathbb{S} -matrix of dimension $X \times Y$* is a mapping $M: X \times Y \rightarrow \mathbb{S}$. The set of all such matrices is denoted by $\mathbb{S}^{X \times Y}$. $M[x, y]$ ($v[x]$) denotes the (x, y) -th entry of M (x -th entry of v). Furthermore $v \cdot s$ denotes the multiplication of a vector with a scalar s and $v_1 + v_2$ is the componentwise addition.

Given a set V of \mathbb{S} -vectors, a *linear combination* of vectors in V is a vector $v_1 \cdot s_1 + \dots + v_n \cdot s_n$, where $v_1, \dots, v_n \in V$, $s_1, \dots, s_n \in \mathbb{S}$. A subset of \mathbb{S}^X that is closed under linear combinations is called a (*sub*-)semimodule.

Henceforward we will always require *l-monoids* to be completely distributive: this ensures that we have a residuation operation defined as follows.

Definition 2.2. The residuation operation for a completely distributive *l-monoid* \mathbb{L} is defined for all $\ell_1, \ell_2 \in \mathbb{L}$ as $\ell_1 \rightarrow \ell_2 = \bigsqcup \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\}$, also called *residuum* of ℓ_1, ℓ_2 . We extend this to \mathbb{L} -vectors, replacing ℓ_1, ℓ_2 by $v_1, v_2 \in \mathbb{L}^X$.

Example 2.3. Recall \mathbb{T}, \mathbb{M} in Example 2.1. For $\ell_1, \ell_2 \in \mathbb{T}$ we have $\ell_1 \rightarrow \ell_2 = \min\{\ell \in \mathbb{R}_0^+ \cup \{\infty\} \mid \ell_1 + \ell \geq \ell_2\} = \ell_2 \dot{-} \ell_1$ (modified subtraction). For $\ell_1, \ell_2 \in \mathbb{M}$, we have $\ell_1 \rightarrow \ell_2 = \max\{\ell \in [0, 1] \mid \ell_1 \cdot \ell \leq \ell_2\} = \min\{1, \frac{\ell_2}{\ell_1}\}$.

Another example where the residuation operation can be easily characterized is any boolean algebra $(\mathbb{B}, \vee, \wedge, 0, 1)$. For $\ell_1, \ell_2 \in \mathbb{B}$ we have $\ell_1 \rightarrow \ell_2 = \neg \ell_1 \vee \ell_2$.

We will assume that all relevant operations of any semiring under consideration (addition, multiplication, in the case of *l-monoids* residuation) are computable.

³ Completely distributive *l-monoids* are often referred to as *unital quantales*.

⁴ We will sometimes use \min as an infix operator (i.e., $a \min b$).

3 Congruence Closure

As explained in the introduction, the key ingredient for exploiting up-to techniques in Section 4 is an algorithmic procedure to check whether two vectors belong to the congruence closure of a given relation of vectors.

3.1 Problem Statement

Let X be a finite set and let \mathbb{S} be a semiring. A relation $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ is a *congruence* if it is an equivalence and *closed under linear combinations*, that is, for each $(v_1, v'_1), (v_2, v'_2) \subseteq R$ and each scalar $s \in \mathbb{S}$, $(v_1 + v_2, v'_1 + v'_2) \in R$ and $(v_1 \cdot s, v'_1 \cdot s) \in R$. The *congruence closure* $c(R)$ of a relation R over a semiring \mathbb{S} is the smallest congruence $R' \subseteq \mathbb{S}^X \times \mathbb{S}^X$ such that $R \subseteq R'$. Alternatively, two vectors $v, v' \in \mathbb{S}^X$ are in $c(R)$ whenever this can be derived via the rules in Table 1.

$$\begin{array}{c}
\text{(REL)} \quad \frac{v \ R \ w}{v \ c(R) \ w} \quad \text{(REFL)} \quad \frac{}{v \ c(R) \ v} \quad \text{(SYM)} \quad \frac{v \ c(R) \ w}{w \ c(R) \ v} \\
\text{(TRANS)} \quad \frac{u \ c(R) \ v \quad v \ c(R) \ w}{u \ c(R) \ w} \quad \text{(SCA)} \quad \frac{v \ c(R) \ w}{v \cdot s \ c(R) \ w \cdot s} \quad \text{where } s \in \mathbb{S} \\
\text{(PLUS)} \quad \frac{v_1 \ c(R) \ v'_1 \quad v_2 \ c(R) \ v'_2}{v_1 + v_2 \ c(R) \ v'_1 + v'_2}
\end{array}$$

Table 1. Proof rules for the congruence closure

Given a finite $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ and $v, w \in \mathbb{S}^X$, we aim to determine if $(v, w) \in c(R)$.

In [8], Bonchi and Pous presented a procedure to compute the congruence closure for the two-valued boolean semiring $B = \{0, 1\}$. The purpose of this section is to generalise the procedure towards more general semirings, such as rings and l -monoids.

3.2 Congruence Closure for Rings

A simple case to start our analysis is the congruence closure of a ring. It is kind of folklore (see e.g. [22,9]) that a submodules⁵ can be used to represent a congruences. In particular we write $[V]$ to denote the submodule generated by a set of vectors V .

Proposition 3.1. *Let \mathbb{R} be a ring and X be a finite set. Let $R \subseteq \mathbb{R}^X \times \mathbb{R}^X$ be a relation and let $(v, v') \in \mathbb{R}^X \times \mathbb{R}^X$ be a pair of vectors. We construct a generating set for a submodule of \mathbb{R}^X by defining $U_R = \{u - u' \mid (u, u') \in R\}$. Then $(v, v') \in c(R)$ iff $v - v' \in [U_R]$.*

This yields an algorithm for a congruence check whenever we have an algorithm to solve linear equations, e.g. for fields. If the ring is not a field, it might still be possible to embed it into a field. In this case we can solve e.g. the language equivalence problem (Section 4.1) for weighted automata in the field and the results are also valid in the ring. Similarly, the procedure can be used for probabilistic automata which can be seen as weighted automata over the reals.

⁵ A sub-semimodule for a ring is called submodule.

3.3 Congruence Closure for l -Monoids

Rewriting and Normal Forms. Our method to determine if a pair of vectors is in the congruence closure is to employ a rewriting algorithm that rewrites both vectors to a normal form. These coincide iff the vectors are related by the congruence closure.

Definition 3.2 (Rewriting and normal forms). *Let \mathbb{L} be an integral l -monoid and let $R \subseteq \mathbb{L}^X \times \mathbb{L}^X$ be a finite relation.*

We define a set of rewriting rules \mathcal{R} as follows: For each pair of vectors $(v, v') \in R$, we obtain two rewriting rules $v \mapsto v \sqcup v'$ and $v' \mapsto v \sqcup v'$.

A rewriting step works as follows: given a vector v and a rewriting rule $l \mapsto r$, we compute the residuum $l \rightarrow v$ and, provided $v \sqsubseteq (v \sqcup r \cdot (l \rightarrow v))$, the rewriting rule is applicable and v rewrites to $v \sqcup r \cdot (l \rightarrow v)$ (symbolically: $v \rightsquigarrow v \sqcup r \cdot (l \rightarrow v)$). A vector v is in normal form wrt. R , provided there exists no rule that is applicable to v .

Example 3.3. *In order to illustrate how rewriting works, we work in \mathbb{T} , set $X = \{1, 2\}$ (two dimensions) and take the relation $R = \{((\infty, \begin{smallmatrix} 0 \\ 0 \end{smallmatrix}), (\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}, \infty))\} \subseteq \mathbb{T}^2 \times \mathbb{T}^2$, relating the two unit vectors, and the vector $v = (\begin{smallmatrix} \infty \\ 3 \end{smallmatrix})$. This yields a rule $l = (\begin{smallmatrix} \infty \\ 0 \end{smallmatrix}) \mapsto r = (\begin{smallmatrix} 0 \\ 0 \end{smallmatrix})$. We obtain $l \rightarrow v = 3$ and hence $v \rightsquigarrow v \sqcup r \cdot (l \rightarrow v) = (\begin{smallmatrix} \infty \\ 0 \end{smallmatrix}) \min (\begin{smallmatrix} 0 \\ 0 \end{smallmatrix} + 3) = (\begin{smallmatrix} 3 \\ 3 \end{smallmatrix})$.*

It is worth to observe that when \mathbb{L} is the boolean semiring, the above procedure coincides with the one in [8]. The rewriting relation satisfies some simple properties:

Lemma 3.4. (i) *If $v \rightsquigarrow v'$ and $v \sqsubseteq w$, then $v' \sqsubseteq w$ or there exists w' s.t. $w \rightsquigarrow w'$ and $v' \sqsubseteq w'$.*
(ii) *Whenever $v \rightsquigarrow v'$ and w is any vector, there exists a vector u s.t. $v \sqcup w \rightsquigarrow u \sqsupseteq v' \sqcup w$ or $v \sqcup w = v' \sqcup w$.*

We now have to prove the following three statements: (i) Our technique is sound, i.e. whenever two vectors have the same normal form wrt. R , they are in $c(R)$. (ii) Our technique is complete, i.e. whenever two vectors are in $c(R)$, they have the same normal form wrt. R . (iii) Our algorithm to compute normal forms terminates.

We will show (i) and prove that (ii) follows from (iii). Afterwards we will discuss sufficient conditions and examples where (iii) holds.

Theorem 3.5. *Whenever there exists a vector v , such that two vectors v_1, v_2 both rewrite to \bar{v} , i.e., $v_1 \rightsquigarrow^* \bar{v}$, $v_2 \rightsquigarrow^* \bar{v}$, then $(v_1, v_2) \in c(R)$.*

Proof. We will show that if v rewrites to v' via a rule $l \mapsto r$, then $(v, v') \in c(R)$.

Since $l \mapsto r$ is a rewriting rule we have that $l = w$, $r = w \sqcup w'$ for $(w, w') \in R$ or $(w', w) \in R$. In both cases $w = w \sqcup w \sqcap c(R) w \sqcup w'$ due to the definition of congruence closure, using rules (PLUS), (REL) and (REFL), as well as (SYM) in case $(w', w) \in R$. Hence $l \sqcap c(R) r$. This implies that $l \cdot (l \rightarrow v) \sqcap c(R) r \cdot (l \rightarrow v)$ (SCA) and furthermore $v \sqcup l \cdot (l \rightarrow v) \sqcap c(R) v \sqcup r \cdot (l \rightarrow v)$ (PLUS). Since $l \cdot (l \rightarrow v) \sqsubseteq v$ we have $v \sqcup l \cdot (l \rightarrow v) = v$ and hence $v \sqcap c(R) v'$. \square

This concludes the proof of soundness, we will go on proving completeness.

Lemma 3.6. *Assume we have a rewriting system that always terminates. Then the local Church-Rosser property holds. That is whenever $v \rightsquigarrow v_1$ and $v \rightsquigarrow v_2$, there exists a vector v' such that $v_1 \rightsquigarrow^* v'$ and $v_2 \rightsquigarrow^* v'$.*

If a rewriting system terminates and the local Church-Rosser property holds, the system is automatically confluent [12]. In this case, every vector v is associated with a unique normal form, written $\Downarrow_{\mathcal{R}} v$ or simply $\Downarrow v$ where $v \rightsquigarrow^* \Downarrow v \not\rightsquigarrow$.

Furthermore, due to Lemma 3.4.(i) we know that \Downarrow is monotone, i.e., $v \sqsubseteq v'$ implies $\Downarrow v \sqsubseteq \Downarrow v'$. This also implies $\Downarrow(v \sqcup v') \sqsubseteq (\Downarrow v) \sqcup (\Downarrow v')$.

Lemma 3.7. *For all $v \in \mathbb{L}^X$, $\ell \in \mathbb{L}$ we have that if $v \rightsquigarrow v'$, then $v \cdot \ell \rightsquigarrow v' \cdot \ell$ for some $v'' \sqsubseteq v' \cdot \ell$ or $v \cdot \ell = v' \cdot \ell$. In particular, if rewriting terminates, we have $(\Downarrow v) \cdot \ell \sqsubseteq \Downarrow(v \cdot \ell)$.*

Now we have all the necessary ingredients to show that the technique is complete, provided the computation of a normal form terminates.

Theorem 3.8. *Assume that rewriting terminates. If $v \in c(R)$ then $\Downarrow v = \Downarrow v'$.*

Termination. One technique to prove termination is given in Corollary 3.10: it suffices to show that the supremum of all the elements reachable via \rightsquigarrow is included in the congruence class. First we need the following result.

Proposition 3.9. *If $v \in c(R)$ \bar{v} , then $v \rightsquigarrow^* v'$ where $v' \sqsubseteq v \sqcup \bar{v}$.*

Now take $\bar{v} = \bigsqcup \{\hat{v} \mid v \rightsquigarrow^* \hat{v}\}$. By the above proposition if $v \in c(R)$ \bar{v} , then $v' = \bar{v}$ and $v \rightsquigarrow^* \bar{v}$. Since \rightsquigarrow is irreflexive, $\bar{v} \not\rightsquigarrow$. If we assume that rule application is fair, we can guarantee that \bar{v} is eventually reached in every rewriting sequence.

Corollary 3.10. *If $v \in c(R)$ $\bigsqcup \{\hat{v} \mid v \rightsquigarrow^* \hat{v}\}$, then the rewriting algorithm terminates, assuming that every rule that remains applicable is eventually applied.*

Termination for Specific l -Monoids. We now study the l -monoid $\mathbb{M} = ([0, 1], \max, \cdot, 0, 1)$ from Example 2.1 and show that the rewriting algorithm terminates for this l -monoid. For the proof we mainly use the pigeon-hole principle and exploit the total ordering of the underlying lattice. Since \mathbb{M} is isomorphic to \mathbb{T} , we obtain termination for the tropical semiring as a corollary.

Theorem 3.11. *The rewriting algorithm terminates for the l -monoids \mathbb{M} and \mathbb{T} .*

These results provide an effective procedure for checking congruence closure over the tropical semiring. We will mainly apply them to weighted automata, but expect that they can be useful to solve other problems. For instance, in Appendix B, we show an interesting connection to the shortest path problem.

Termination for Lattices. We next turn to lattices and give a sufficient condition for termination on lattices. Obviously, rewriting terminates for lattices for which the ascending chain condition holds (i.e., every ascending chain eventually becomes stationary), but one can go beyond that.

In this section, we assume a completely distributive lattice \mathbb{L} and a boolean algebra \mathbb{B} such that the orders of \mathbb{L} and \mathbb{B} , as well as the infima coincide. Suprema need not coincide. Thus, whenever there is ambiguity, we will add the index \mathbb{B} or \mathbb{L} to the operator. For the negation of a given $x \in \mathbb{B}$, we write $\neg x$. One way to obtain such a boolean algebra – in particular one where the suprema coincide as well – is via Funayama’s theorem, see [5]. This embedding is also discussed in Appendix A.2.

We want to show that if \mathbb{L} approximates \mathbb{B} “well enough”, the rewriting algorithm terminates for \mathbb{L} .

Theorem 3.12. *The approximation of an element $\ell \in \mathbb{B}$ in the lattice \mathbb{L} is defined as $\llbracket \ell \rrbracket = \bigsqcup_{\mathbb{L}} \{\ell' \in \mathbb{L} \mid \ell' \leq \ell\}$.*

Let \mathcal{R} be a rewriting system for vectors in \mathbb{L}^X . Whenever the set $L(l, x) = \{\ell \in \mathbb{L} \mid \llbracket \neg l[x] \rrbracket \sqsubseteq \ell \sqsubseteq \neg l[x]\}$ is finite for all rules $(l \mapsto r) \in \mathcal{R}$ and all $x \in X$, rewriting terminates.

Note that $\llbracket \neg \ell \rrbracket = \llbracket \ell \rightarrow_{\mathbb{B}} 0 \rrbracket = \ell \rightarrow_{\mathbb{L}} 0$ (Lemma A.4). Hence the theorem says that there should be only finitely many elements between the negation of an element in the lattice and the negation of the same element in the boolean algebra. As a simple corollary we obtain that the rewriting algorithm terminates for all boolean algebras.

4 Up-To Techniques for Weighted Automata

In this section we present applications of our congruence closure method. More specifically, we investigate weighted automata and present up-to techniques both for the language equivalence and the inclusion problem, which are variants of the efficient up-to based algorithm presented in [8]. For the tropical semiring we also give a procedure for solving the threshold problem, based on the language inclusion algorithm.

4.1 Language Equivalence for Weighted Automata

We turn our attention towards weighted automata and their languages.

A *weighted automaton* over the semiring \mathbb{S} and alphabet A is a triple (X, o, t) where X is a finite set of states, $t = (t_a: X \rightarrow \mathbb{S}^X)_{a \in A}$ is an A -indexed set of transition functions and $o: X \rightarrow \mathbb{S}$ is the output function. Intuitively $t_a(x)(y) = s$ means that the states x can make a transition to y with letter $a \in A$ and weight $s \in \mathbb{S}$ (sometimes written as $x \xrightarrow{a, s} y$). The functions t_a can be represented as $X \times X$ -matrices with values in \mathbb{S} and o as a row vector of dimension X . Given a vector $v \in \mathbb{S}^X$, we use $t_a(v)$ to denote the vector obtained by multiplying the matrix t_a by v and $o(v)$ to denote the scalar in \mathbb{S} obtained by multiplying the row vector o by the column vector v .

A *weighted language* is a function $\varphi: A^* \rightarrow \mathbb{S}$, where A^* is the set of all words over A . We will use ε to denote the empty word and aw the concatenation of a letter $a \in A$ with the word $w \in A^*$. Every weighted automaton is associated with a function $\llbracket - \rrbracket: \mathbb{S}^X \rightarrow \mathbb{S}^{A^*}$ mapping each vector into its *accepted language*. For all $v \in \mathbb{S}^X$, $a \in A$ and $w \in A^*$, this is defined as

$$\llbracket v \rrbracket(\varepsilon) = o(v) \quad \llbracket v \rrbracket(aw) = \llbracket t_a(v) \rrbracket(w).$$

$$\underline{\text{HKC}}(v_1, v_2)$$

```

(1)  $R := \emptyset$ ;  $todo := \emptyset$ 
(2) insert  $(v_1, v_2)$  into  $todo$ 
(3) while  $todo$  is not empty do
  (3.1) extract  $(v'_1, v'_2)$  from  $todo$ 
  (3.2) if  $(v'_1, v'_2) \in c(R)$  then continue
  (3.3) if  $o(v'_1) \neq o(v'_2)$  then return false
  (3.4) for all  $a \in A$ ,
        insert  $(t_a(v'_1), t_a(v'_2))$  into  $todo$ 
  (3.5) insert  $(v'_1, v'_2)$  into  $R$ 
(4) return true

```

Fig. 1. Algorithm to check the equivalence of vectors $v_1, v_2 \in \mathbb{S}^X$ for a weighted automata (X, o, t) .

Two vectors $v_1, v_2 \in \mathbb{S}^X$ are called *language equivalent*, written $v_1 \sim v_2$ iff $\llbracket v_1 \rrbracket = \llbracket v_2 \rrbracket$.⁶ The problem of checking language equivalence in weighted automata for an arbitrary semiring is undecidable: for the tropical semiring this was shown by Krob in [18]; the proof was later simplified in [3]. However, for several semirings the problem is decidable, for instance for all (complete and distributive) lattices. For finite non-deterministic automata, i.e., automata weighted over the boolean semiring, Bonchi and Pous introduced in [8] the algorithm HKC. The name stems from the fact that the algorithm extends the procedure of Hopcroft and Karp [16] with congruence closure.

Fig. 1 shows the extension of HKC to weighted automata over an arbitrary semiring: the code is the same as the one in [8], apart from the fact that, rather than exploring sets of states, the algorithm works with vectors in \mathbb{S}^X . The check at step (3.2) can be performed with the procedures discussed in Section 3.

Below we prove that the algorithm is sound and complete, but termination can fail in two ways: either the check at step (3.2) does not terminate, or the while loop at step (3) does not. For the tropical semiring we have seen that the check at step (3.2) can be effectively performed by rewriting (Theorem 3.11). Therefore, due to Krob's undecidability result, the while loop at step (3) may not terminate. For (distributive) lattices, we have shown termination of rewriting under some additional constraints (Theorem 3.12); moreover the loop at (3) will always terminate, because from a given finite set of lattice elements only finitely many lattice elements can be constructed using infimum and supremum [17].

To prove soundness of HKC, we introduce the notions of simulation and bisimulation up-to. Let $Rel_{\mathbb{S}^X}$ be the complete lattice of relations over \mathbb{S}^X and $b_1 : Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$ be the monotone map defined for all $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ as

$$b_1(R) = \{(v_1, v_2) \mid o(v_1) = o(v_2) \text{ and for all } a \in A, (t_a(v_1), t_a(v_2)) \in R\}$$

⁶ The accepted notions of language and language equivalence can be given for states rather than for vectors by assigning to each state $x \in X$ the corresponding unit vector $e_x \in \mathbb{S}^X$. On the other hand, when weighted automata are given with an initial vector i – which is often the case in literature – one can define the language of an automaton as $\llbracket i \rrbracket$.

Definition 4.1. A relation $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ is a b_1 -simulation if $R \subseteq b_1(R)$, i.e., for all $(v_1, v_2) \in R$: (i) $o(v_1) = o(v_2)$; (ii) for all $a \in A$, $(t_a(v_1), t_a(v_2)) \in R$.

For a monotone map $f: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$, a b_1 -simulation up-to f is a relation R such that $R \subseteq b_1(f(R))$.

It is easy to show (see e.g. [20]) that b_1 -simulation provides a sound and complete proof technique for \sim . On the other hand, not all functions f can be used as sound up-to techniques. HKC exploits the monotone function $c: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$ mapping each relation R to its congruence closure $c(R)$.

Proposition 4.2. Let $v_1, v_2 \in \mathbb{S}^X$. It holds that $v_1 \sim v_2$ iff there exists a b_1 -simulation R such that $(v_1, v_2) \in R$ iff there exists a b_1 -simulation up-to c R such that $(v_1, v_2) \in R$.

With this result, it is easy to prove the correctness of the algorithm.

Theorem 4.3. Whenever HKC terminates, it returns true iff $\llbracket v_1 \rrbracket = \llbracket v_2 \rrbracket$.

Proof. Observe that $R \subseteq b_1(c(R) \cup \text{todo})$ is an invariant for the while loop at step (3).

If HKC returns *true* then *todo* is empty and thus $R \subseteq b_1(c(R))$, i.e., R is a b_1 -simulation up-to c . By Proposition 4.2, $v_1 \sim v_2$.

Whenever HKC returns *false*, it encounters a pair $(v'_1, v'_2) \in \text{todo}$ such that $o(v'_1) \neq o(v'_2)$. Observe that for all pairs $(v'_1, v'_2) \in \text{todo}$, there exists a word $w = a_1 a_2 \dots a_n \in A^*$ such that $v'_1 = t_{a_n}(\dots t_{a_2}(t_{a_1}(v_1)))$ and $v'_2 = t_{a_n}(\dots t_{a_2}(t_{a_1}(v_2)))$. Therefore $\llbracket v_1 \rrbracket(w) = \llbracket v'_1 \rrbracket(\varepsilon) = o(v'_1) \neq o(v'_2) = \llbracket v'_2 \rrbracket(\varepsilon) = \llbracket v_2 \rrbracket(w)$. \square

4.2 Language Inclusion

Whenever a semiring \mathbb{S} carries a partial order \sqsubseteq , one can be interested in checking language inclusion of the states of a weighted automata (X, o, t) . More generally, given $v_1, v_2 \in \mathbb{S}^X$, we say that the language of v_1 is included in the language of v_2 (written $v_1 \sqsubseteq v_2$) iff $\llbracket v_1 \rrbracket(w) \sqsubseteq \llbracket v_2 \rrbracket(w)$ for all $w \in A^*$.

The algorithm HKC can be slightly modified to check language inclusion, resulting in algorithm HKP: steps (3.2) and (3.3) are replaced by

```
(3.2)  if  $(v'_1, v'_2) \in p(R)$  then continue
(3.3)  if  $o(v'_1) \not\sqsubseteq o(v'_2)$  then return false
```

where $p: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$ is the monotone function assigning to each relation R its pre-congruence closure $p(R)$.

The precongruence closure is defined as the closure of R under \sqsubseteq , transitivity and linear combination. That is, in the rules of Table 1 $c(R)$ is replaced by $p(R)$, rule (SYM) is removed and rule (REFL) is replaced by rule (ORD) on the right.

$$(\text{ORD}) \frac{v \sqsubseteq v'}{v \ p(R) \ v'}$$

The soundness of the modified algorithm can be proved in the same way as for HKC by replacing c by p and b_1 by $b_2: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$ defined for all $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ as

$$b_2(R) = \{(v_1, v_2) \mid o(v_1) \sqsubseteq o(v_2) \text{ and for all } a \in A, (t_a(v_1), t_a(v_1)) \in R\}.$$

However, the soundness of up-to reasoning is guaranteed only if \sqsubseteq is a precongruence, that is $p(\sqsubseteq)$ is contained in \sqsubseteq .

Theorem 4.4. *Let \mathbb{S} be a semiring equipped with a precongruence \sqsubseteq . Whenever $\text{HKP}(v_1, v_2)$ terminates, it returns true if and only if $v_1 \sqsubseteq v_2$.*

In order for HKP to be effective, we need a procedure to compute p . When \mathbb{S} is an integral l -monoid, we can check $(v, v') \in p(R)$ via a variation of the congruence check, using a rewriting system as in Section 3.3.

Proposition 4.5. *Let \mathbb{L} be an integral l -monoid and let $R \subseteq \mathbb{L}^X \times \mathbb{L}^X$ be a relation.*

The set of rules \mathcal{R} is defined as $\{v' \mapsto v \sqcup v' \mid (v, v') \in R\}$.⁷ Rewriting steps are defined as in Definition 3.2. If the rewriting algorithm terminates, then for all $v, v' \in \mathbb{L}^X$, $(v, v') \in p(R)$ iff $\Downarrow v' \geq v$ (where, as usual, $\Downarrow v'$ denotes the normal form of v').

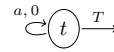
Observe that Theorems 3.11 and 3.12 guarantee termination for certain specific l -monoids. In particular, termination for the tropical semiring will be pivotal henceforward.

4.3 Threshold Problem for Automata over the Tropical Semiring

Language inclusion for weighted automata over the tropical semiring is not decidable, because language equivalence is not. However, the algorithm that we have introduced in the previous section can be used to solve the so called *threshold problem* over the tropical semiring of natural numbers $(\mathbb{N}_0 \cup \{\infty\}, \min, +, \infty, 0)$. The problem is to check whether for a given threshold $T \in \mathbb{N}_0$, a vector of states of a weighted automaton $v \in (\mathbb{N}_0 \cup \{\infty\})^X$ satisfies the threshold T , i.e. $\llbracket v \rrbracket(w) \leq T$ for all $w \in A^*$.

Note that this problem is also known as the *universality problem*: universality for non-deterministic automata can be easily reduced to it, by taking weight 0 for each transition and setting $T = 0$ for the threshold.

This problem – which is known to be PSPACE-complete [3] – can be reduced to language inclusion by adding a new state t with output $o(t) = T$ and a 0 self-loop for each letter $a \in A$. Then we check whether the language of v includes the language of the unit vector e_t .



It is worth to note that in $(\mathbb{N}_0 \cup \{\infty\}, \min, +, \infty, 0)$ the ordering \sqsubseteq is actually \geq , the reversed ordering on natural numbers. Therefore to solve the threshold problem, we need to check $e_t \sqsubseteq v$.

The reader can easily concoct an example where HKP may not terminate. However, it has already been observed in [3] that it is a sound reasoning technique to replace every vector entry larger than T by ∞ . To formalise this result, we will first introduce an abstraction mapping \mathcal{A} and then state our modified algorithm:

Definition 4.6. *Let a threshold $T \in \mathbb{N}_0$ be given. We define the abstraction $\mathcal{A} : \mathbb{N}_0 \cup \{\infty\} \rightarrow \mathbb{N}_0 \cup \{\infty\}$ according to $\mathcal{A}(s) = s$ if $s \leq T$ and $\mathcal{A}(s) = \infty$ otherwise. The definition extends elementwise to vectors in $(\mathbb{N}_0 \cup \{\infty\})^X$.*

⁷ Whenever $v \leq v'$, the rule can be omitted, since it is never applicable.

With this definition, we call HKP_A , the algorithm obtained from HKP by replacing step (3.4) with the following:

(3.4) **for all** $a \in A$
 insert $(t_a(v'_1), \mathcal{A}(t_a(v'_2)))$ into *todo*

Now to check whether a certain vector v satisfies the threshold of T , it is enough to run $\text{HKP}_A(e_t, v)$ where e_t is the unit vector for t as defined above.

The soundness of the proposed algorithm can be shown in essentially the same way as for HKP but using a novel up-to technique to take care of the abstraction \mathcal{A} (see Appendix A.3). For the completeness, we need the following additional result.

Lemma 4.7. *For all vectors $v \in (\mathbb{N}_0 \cup \{\infty\})^X$ it holds that (i) $\mathcal{A}(t_a(\mathcal{A}(v))) = \mathcal{A}(t_a(v))$; (ii) $\mathcal{A}(o(\mathcal{A}(v))) = \mathcal{A}(o(v))$.*

Theorem 4.8. *$\text{HKP}_A(e_t, v_1)$ always terminates. Moreover $\text{HKP}_A(e_t, v_1)$ returns true iff $\llbracket v_1 \rrbracket(w) \leq T$ for all $w \in A^*$.*

4.4 Exploiting Similarity

For checking language inclusion of non-deterministic automaton it is often convenient to precompute a *similarity* relation that allows to immediately skip some pairs of states [1]. This idea can be readapted to weighted automata over an l -monoid by using the following notion.

Definition 4.9. *Let (X, o, t) be a weighted automaton. A relation $R \subseteq \mathbb{S}^X \times \mathbb{S}^X$ on unit vectors is called a simulation relation whenever for all $(v, v') \in R$ (i) $o(v) \sqsubseteq o(v')$; (ii) for all $a \in A$, there exists a pair (u, u') that is a linear combination of vector pairs in R and furthermore $t_a(v) \sqsubseteq u, u' \sqsubseteq t_a(v')$.*

Similarity, written \preceq , is the greatest simulation relation.

Lemma 4.10. *Simulation implies language inclusion, i.e. \preceq is included in \sqsubseteq .*

Similarity over an l -monoid can be computed with the algorithm in Fig. 2. Even though the relation is not symmetric, the method is conceptually close to the traditional partition refinement algorithm to compute bisimilarity. Starting from the cross-product of all states, the algorithm first eliminates all pairs of states where the first state does not have a smaller-or-equal output than the second one and then continuously removes all pairs of states that do not meet the second requirement for a simulation relation, until the relation does not change anymore.

Lemma 4.11. *SIM computes \preceq .*

Lemma 4.12. *The runtime complexity of SIM when applied to an automaton over state set X and alphabet A is polynomial, assuming constant time complexity for all semiring operations (supremum, multiplication, residuation).*

Once \preceq is known, it can be exploited by HKP and HKP_A . To be completely formal in the proofs, it is convenient to define two novel algorithm – called HKP' and HKP'_A – which are obtained from HKP and HKP_A by replacing step (3.2) by

$$\underline{\text{SIM}}(X, o, t)$$

```

(1)   $R := \{(v, v') \in S^X \times S^X \mid v, v' \text{ are unit vectors}\}$ 
(2)   $R' := \emptyset$ 
(3)  for all  $(v, v') \in R$ 
    (3.1) if  $o(v) \not\sqsubseteq o(v')$  then  $R := R \setminus \{(v, v')\}$ 
(4)  while  $R \neq R'$ 
    (4.1)  $R' := R$ 
    (4.2) for all  $a \in A$ 
        (4.2.1) for all  $(v, v') \in R$ 
            (4.2.1.1)  $u := \bigsqcup \{v_1 \cdot (v_2 \rightarrow t_a(v')) \mid (v_1, v_2) \in R\}$ 
            (4.2.1.2) if  $t_a(v) \not\sqsubseteq u$  then  $R := R \setminus \{(v, v')\}$ 
(5)  return  $R$ 

```

Fig. 2. Algorithm to compute similarity (\preceq) for a weighted automaton (X, o, t) .

```

(3.2) if  $(v'_1, v'_2) \in p'(R)$  then continue

```

where $p'(R)$ is defined for all relations R as $p'(R) = p(R \cup \preceq)$. The following two results state the correctness of the two algorithms.

Lemma 4.13. *Let \mathbb{S} be a semiring equipped with a precongruence \sqsubseteq . Whenever $\text{HKP}'(v_1, v_2)$ terminates, it returns true iff $v_1 \preceq v_2$.*

Lemma 4.14. *$\text{HKP}'_A(e_t, v_1)$ always terminates. Moreover $\text{HKP}'_A(e_t, v_1)$ returns true iff $\llbracket v_1 \rrbracket(w) \leq T$ for all $w \in A^*$.*

4.5 An Exponential Pruning

To illustrate the benefits of up-to techniques, we show an example where HKP'_A exponentially prunes the exploration space by exploiting the technique p' . We compare HKP'_A against ABK in Fig. 3, that can be thought as an adaptation of the algorithm proposed in [3] to the notation used in this paper.

Consider the family of automata over the tropical semiring in Fig. 4 and assume that $T = n$. By taking as initial vector $e_x \sqcup e_y$ (i.e., the vector mapping x and y to 0 and all other states to ∞), the automaton clearly does not respect the threshold, but this can be observed only for words longer than n .

First, for ABK the runtime is exponential. This happens, since every word up to length n produces a different weight vector. For a word w of length m state x_i has weight m iff the i -last letter of the word is a , similarly state y_i has weight m iff the i -last letter is b . All other weights are ∞ . For instance, the weights for word aab are given below.

x	x_1	x_2	x_3	x_4	\dots	y	y_1	y_2	y_3	y_4	\dots
3	∞	3	3	∞	\dots	3	3	∞	∞	∞	\dots

ABK(v_0)

```

(1)  todo := { $v_0$ }
(2)   $P := \emptyset$ 
(3)  while todo  $\neq \emptyset$ 
      (3.1) extract  $v$  from todo
      (3.2) if  $v \in P$  then continue
      (3.3) if  $o(v) \not\leq T$  then return false
      (3.4) for all  $a \in A$  insert  $\mathcal{A}(t_a(v))$  into todo
      (3.5) insert  $v$  into  $P$ 
(4)  return true

```

Fig. 3. Algorithm to check whether a vector v_0 of a weighted automata (X, o, t) satisfies the threshold $T \in \mathbb{N}_0$

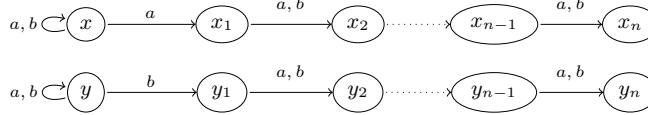


Fig. 4. Examples where HKP'_A exponentially improves over ABK. Output weight is always 0, transition weight is always 1.

Now we compare with HKP'_A . Observe that $x_i \preceq x$, $y_i \preceq y$ for all i . (Remember that since the order is reversed, a lower weight simulates a higher weight.) Hence, we obtain rewriting rules that allow to replace an ∞ -entry in x_i and y_i by m for all i . (Since both entries x and y are m , we can always apply this rule.) In the example above this leads to a vector where every entry is 3.

Hence it turns out that for all words of the same length, the corresponding vectors are all in the precongruence relation with each other – as they share the same normal form – and we only have to consider exactly one word of each length. Therefore, only linearly many words are considered and the runtime is polynomial.

5 Runtime Results for the Threshold Problem

We now discuss runtime results for the threshold problem for weighted automata over the tropical semiring of the natural numbers. We compare the following three algorithms: the algorithm without up-to technique (ABK) algorithm in Fig. 3, the algorithm that works up-to precongruence (HKP_A), explained in Section 4.3, and the algorithm that additionally exploits pre-computed similarity (HKP'_A), introduced in Section 4.4. This precomputation step is relatively costly and is included in the runtime results below.

We performed the following experiment: for certain values of $|X|$ (size of state set) and of T (threshold) we generated random automata. The alphabet size was randomly chosen between 1 and 5. For each pair of states and alphabet symbol, the probability of having an edge with finite weight is 90%. (We chose this high number, since otherwise

the threshold is almost never respected and the algorithms return false almost immediately due to absence of a transition for a given letter. With our choice instead, the algorithms need many steps and the threshold is satisfied in 14% of the cases.) In case the weight is different from ∞ , a random weight from the set $\{0, \dots, 10\}$ is assigned.

For each pair $(|X|, T)$ we generated 1000 automata. The runtime results can be seen in Table 2. We considered the 50%, 90% and 99% percentiles: the 50% percentile is the median and the 90% percentile means that 90% of the runs were faster and 10% slower than the time given in the respective field. Analogously for the 99% percentile.

Apart from the runtime we also measured the size of the relation R (or P in the case of ABK) and the size of the similarity \preceq (in case of HKP'_A). The program was written in C# and executed on an Intel Core 2 Quad CPU Q9550 at 2.83 GHz with 4 GB RAM, running Windows 10.

First note that, as expected, HKP_A and HKP'_A always produce much smaller relations than ABK . However, they introduce some overhead, due to rewriting for checking $p(R)$, and due to the computation of similarity, which is clearly seen for the 50% percentile. However, if we look at the larger parameters and at the 90% and 99% percentiles (which measure the worst-case performance), HKP_A and HKP'_A gain the upper hand in terms of runtime.

Note also that while in the example above similarity played a large role, this is not the case for the random examples. Here similarity (not counting the reflexive pairs) is usually quite small. This means that similarity does not lead to savings, only in very few cases does the size of R decrease for HKP'_A . But this also means that the computation of \preceq is not very costly and hence the runtime of HKP_A is quite similar to the runtime of HKP'_A . We believe that for weighted automata arising from concrete problems, the similarity relation will usually be larger and promise better runtimes. Note also that similarity is independent of the initial vector and the threshold and if one wants to ask several threshold questions for the same automaton, it has to be computed only once.

6 Conclusion and Future Work

In this work, we have investigated up-to techniques for weighted automata, including methods to determine the congruence closure for semimodules.

Related work: Related work on up-to techniques has already been discussed in the introduction. For the language equivalence problem for weighted automata we are mainly aware of the algorithm presented in [4], which is a partition refinement algorithm and which already uses a kind of up-to technique: it can eliminate certain vectors which arise as linear combinations of other vectors. The paper [23] considers simulation for weighted automata, but not in connection to up-to techniques.

Congruence closure for term rewriting has been investigated in [10].

Our examples mainly involved the tropical semiring (and related semirings). Hence there are relations to work by Aceto et al. [2] who presented an equational theory for the tropical semiring and related semirings, as well as Gaubert et al. [14] who discuss several reasons to be interested in the tropical semiring and present solution methods for several types of linear equation systems.

		Runtime (millisec.)			Size of R/P			Size of \prec		
(X , T)	algo	50%	90%	99%	50%	90%	99%	50%	90%	99%
(3,10)	HKP'_A	2	8	20	5	14	33	0	2	4
	HKP_A	1	3	14	5	14	34	-	-	-
	ABK	1	3	13	6	28	92	-	-	-
(3,15)	HKP'_A	3	17	127	11	34	100	0	2	4
	HKP_A	2	16	126	11	34	100	-	-	-
	ABK	2	17	90	18	119	373	-	-	-
(3,20)	HKP'_A	6	65	393	18	70	174	0	2	4
	HKP_A	4	64	466	18	71	192	-	-	-
	ABK	5	79	315	55	364	825	-	-	-
(6,10)	HKP'_A	21	227	1862	18	106	302	0	2	12
	HKP_A	8	217	1858	19	106	302	-	-	-
	ABK	9	286	2045	40	693	2183	-	-	-
(6,15)	HKP'_A	90	2547	12344	65	353	750	0	2	11
	HKP_A	84	2560	12328	65	353	750	-	-	-
	ABK	88	4063	20987	346	3082	7270	-	-	-
(6,20)	HKP'_A	239	7541	59922	111	589	1681	0	3	11
	HKP_A	234	7613	60360	111	589	1681	-	-	-
	ABK	253	16240	103804	702	6140	14126	-	-	-
(9,10)	HKP'_A	274	9634	73369	98	582	1501	0	3	21
	HKP_A	236	9581	72833	99	582	1501	-	-	-
	ABK	232	17825	99332	536	6336	14956	-	-	-
(9,15)	HKP'_A	1709	71509	301033	256	1517	3319	0	3	19
	HKP_A	1681	70587	301018	256	1517	3319	-	-	-
	ABK	919	112323	515386	1436	14889	28818	-	-	-
(9,20)	HKP'_A	3885	168826	874259	407	2347	5086	0	3	20
	HKP_A	3838	168947	872647	407	2347	5086	-	-	-
	ABK	1744	301253	1617813	2171	22713	48735	-	-	-
(12,10)	HKP'_A	1866	93271	560824	247	1586	3668	0	7	31
	HKP_A	1800	92490	560837	251	1586	3668	-	-	-
	ABK	1067	189058	889949	1342	18129	37387	-	-	-
(12,15)	HKP'_A	5127	363530	1971541	423	3001	6743	0	7	35
	HKP_A	5010	362908	1968865	423	3001	6743	-	-	-
	ABK	1418	509455	2349335	1672	27225	55627	-	-	-
(12,20)	HKP'_A	15101	789324	3622374	744	4489	9027	0	6	32
	HKP_A	15013	787119	3623393	744	4489	9027	-	-	-
	ABK	4169	1385929	4773543	3297	43756	80712	-	-	-

Table 2. Runtime results on randomly generated automata

Future work: As we have seen in the experiments on the threshold problem, our techniques greatly reduce the size of the relations. However, the reduction in runtime is less significant, which is due to the overhead for the computation of similarity and the rewriting procedure. There is still a substantial improvement for the worst-case running times (90% and 99% percentiles). So far, the algorithms, especially algorithm SIM for computing similarity, are not very sophisticated and we believe that there is further potential for optimization.

Acknowledgements: We would like to thank Paweł Sobociński and Damien Pous for interesting discussions on the topic of this paper. Furthermore we express our thanks to Issai Zaks for his help with the runtime results.

References

1. Parosh A. Abdulla, Yu-Fang Chen, Lukáš Holík, and Tomáš Vojnar. When simulation meets antichains (on checking language inclusion of NFAs). In *Proc. of TACAS '10*, pages 158–174. Springer, 2010. LNCS 6015.
2. Luca Aceto, Zoltán Ésik, and Anna Ingólfssdóttir. Equational theories of tropical semirings. *Theoretical Computer Science*, 298(3):417 – 469, 2003. Foundations of Software Science and Computation Structures.
3. Shaull Almagor, Udi Boker, and Orna Kupferman. What’s decidable about weighted automata? In *Proc. of ATVA '11*, pages 482–491. Springer, 2011. LNCS 6996.
4. Mariel-Pierre Béal, Sylvain Lombardy, and Jacques Sakarovitch. Conjugacy and equivalence of weighted automata and functional transducers. In *Prof. of CSR '06*, pages 58–69. Springer, 2006. LNCS 3967.
5. Guram Bezhanishvili, David Gabelaia, and Mamuka Jibladze. Funayama’s theorem revisited. *Algebra universalis*, 70(3):271–286, 2013.
6. Filippo Bonchi, Marcello M. Bonsangue, Michele Boreale, Jan J. M. M. Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, 2012.
7. Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In Thomas A. Henzinger and Dale Miller, editors, *Proc. of CSL-LICS '14*, pages 20:1–20:9. ACM, 2014.
8. Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. of POPL '13*, pages 457–468. ACM, 2013.
9. Michele Boreale. Weighted bisimulation in linear algebraic form. In *Proc. of CONCUR '09*, pages 163–177. Springer, 2009. LNCS 5710.
10. David Cyrluk, Patrick Lincoln, and Natarajan Shankar. On Shostak’s decision procedure for combinations of theories. In *Proc. of CADE-13*, pages 463–477. Springer-Verlag, 1996.
11. Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 2002.
12. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Formal Models and Semantics, Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, 1990.
13. Laurent Doyen and Jean-François Raskin. Antichain Algorithms for Finite Automata. In *Proc. of TACAS*, pages 2–22. Springer, 2010. LNCS 6015.
14. Stéphane Gaubert and Max Plus. Methods and applications of (max,+) linear algebra. In *Proc. of STACS '97*, pages 261–282. Springer Berlin Heidelberg, 1997. LNCS 1200.
15. Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. Comput.*, 145(2):107–152, 1998.
16. John E. Hopcroft and Richard M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report TR 114, Cornell University, 1971.
17. Barbara König and Sebastian Küpper. A generalized partition refinement algorithm, instantiated to language equivalence checking for weighted automata. *Soft Computing*, pages 1–18, 2016.
18. Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

19. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
20. Damien Pous and Davide Sangiorgi. Enhancements of the coinductive proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2011.
21. Jurriaan Rot, Filippo Bonchi, Marcello Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced coalgebraic bisimulation. *Mathematical Structures in Computer Science*, pages 1–29, 2015.
22. Eugene W. Stark. On behaviour equivalence for probabilistic i/o automata and its relationship to probabilistic bisimulation. *Journal of Automata, Languages and Combinatorics*, 8(2):361–395, 2003.
23. Natsuki Urabe and Ichiro Hasuo. Generic forward and backward simulations III: Quantitative simulations by matrices. In *Proc. of CONCUR '14*, pages 451–466. Springer, 2014. LNCS/ARCoSS 8704.
24. Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. of CAV '06*, pages 17–30. Springer, 2006. LNCS 4144.

A Proofs

Here we give proofs for all lemmas and propositions where we have omitted the proofs in the article.

A.1 Preliminaries

We will prove some properties l -monoids and residuation which will play an important role in proving our main results.

Lemma A.1. *Let $(\mathbb{L}, \sqcup, \cdot, 0, 1)$ be an l -monoid and v, v' be n -dimensional \mathbb{L} -vectors. Then it holds that*

$$v \rightarrow v' = \prod \{v[i] \rightarrow v'[i] \mid 1 \leq i \leq n\}$$

Proof. We define $(v \Rightarrow v') := \prod \{v[i] \rightarrow v'[i] \mid 1 \leq i \leq n\}$.

First we will show that $v \Rightarrow v' \sqsubseteq v \rightarrow v'$. In order to prove this, we will show that $v \cdot (v \Rightarrow v') \leq v'$, i.e. $(v \Rightarrow v') \in \{\ell \in \mathbb{L} \mid v \cdot \ell \sqsubseteq v'\}$:

$$\begin{aligned} v \cdot (v \Rightarrow v') &= \begin{pmatrix} v[1] \\ v[2] \\ \vdots \\ v[n] \end{pmatrix} \cdot (v \Rightarrow v') = \begin{pmatrix} v[1] \cdot (v \Rightarrow v') \\ v[2] \cdot (v \Rightarrow v') \\ \vdots \\ v[n] \cdot (v \Rightarrow v') \end{pmatrix} \sqsubseteq \begin{pmatrix} v[1] \cdot (v[1] \rightarrow v'[1]) \\ v[2] \cdot (v[2] \rightarrow v'[2]) \\ \vdots \\ v[n] \cdot (v[n] \rightarrow v'[n]) \end{pmatrix} \\ &\sqsubseteq \begin{pmatrix} v'[1] \\ v'[2] \\ \vdots \\ v'[n] \end{pmatrix} = v' \end{aligned}$$

Next we need to show that $v \Rightarrow v' \sqsupseteq v \rightarrow v'$. It suffices to show that $v \Rightarrow v'$ is an upper bound of the set $\{\ell \in \mathbb{L} \mid v \cdot \ell \sqsubseteq v'\}$. Since $v \Rightarrow v'$ is the greatest lower bound of

$\{v[i] \rightarrow v'[i] \mid 1 \leq i \leq n\}$, it is enough to show that every element of the first set and every element of the second set are in relation. Hence take $\ell \in \mathbb{L}$ with $v \cdot \ell \sqsubseteq v'$ and an index i . Since $v \cdot \ell \sqsubseteq v'$ (componentwise), it holds that $v[i] \cdot \ell \sqsubseteq v'[i]$ for every i . Hence $\ell \sqsubseteq v[i] \sqsubseteq v'[i]$.

□

Lemma A.2. *In an l -monoid \mathbb{L} ,*

- (i) *Whenever $\ell_1 \sqsubseteq \ell_2$ it follows that $\ell \cdot \ell_1 \sqsubseteq \ell \cdot \ell_2$ and $\ell_1 \cdot \ell \sqsubseteq \ell_2 \cdot \ell$ for all l -monoid elements ℓ, ℓ_1, ℓ_2 .*
- (ii) *For all l -monoid vectors $v, v' \in \mathbb{L}^Y$ and matrices $M \in \mathbb{L}^{X \times Y}$ it holds that $v \sqsubseteq v'$ implies $Mv \sqsubseteq Mv'$.*
- (iii) *if \mathbb{L} is integral, $\ell \cdot \ell' \sqsubseteq \ell'$ and $\ell \cdot \ell' \sqsubseteq \ell$.*
- (iv) *For all $\ell_1, \ell_2, \ell_3 \in \mathbb{L}$, we have $(\ell_1 \rightarrow \ell_2) \cdot \ell_3 \sqsubseteq \ell_1 \rightarrow (\ell_2 \cdot \ell_3)$.*
- (v) *If $\ell_2 \sqsubseteq \ell_3$ it follows that $(\ell_1 \rightarrow \ell_2) \sqsubseteq (\ell_1 \rightarrow \ell_3)$.*
- (vi) *It holds that $\ell_3 \sqsubseteq \ell_1 \rightarrow \ell_2 \iff \ell_1 \cdot \ell_3 \sqsubseteq \ell_2$.*
- (vii) *It holds that $\ell_1 \rightarrow (\ell_2 \sqcup \ell_3) \sqsupseteq (\ell_1 \rightarrow \ell_2) \sqcup (\ell_1 \rightarrow \ell_3)$.*
- (viii) *It holds that $\ell_1 \cdot (\ell_1 \rightarrow \ell_2) \sqsubseteq \ell_2$*

Proof. (i)

$$\ell \cdot \ell_2 = \ell \cdot (\ell_1 \sqcup \ell_2) = \ell \cdot \ell_1 \sqcup \ell \cdot \ell_2 \sqsupseteq \ell \cdot \ell_1$$

and

$$\ell_2 \cdot \ell = (\ell_1 \sqcup \ell_2) \cdot \ell = \ell_1 \cdot \ell \sqcup \ell_2 \cdot \ell \sqsupseteq \ell_1 \cdot \ell.$$

(ii) The second part follows directly, because

$$(Mv)[i] = \bigsqcup_{j \in Y} M[i, j] \cdot v[j] \sqsubseteq \bigsqcup_{v[j] \sqsubseteq v'[j]} M[i, j] \cdot v'[j] = (Mv')[i]$$

(iii) This follows directly from monotonicity, $\ell \cdot \ell' \sqsubseteq \top \cdot \ell' = 1 \cdot \ell' = \ell'$ and $\ell \cdot \ell' \sqsubseteq \ell \cdot \top = \ell \cdot 1 = \ell$.

(iv) We first compute:

$$(\ell_1 \rightarrow \ell_2) \cdot \ell_3 = \bigsqcup \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \cdot \ell_3 = \bigsqcup \{\ell \cdot \ell_3 \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\}$$

and:

$$\ell_1 \rightarrow (\ell_2 \cdot \ell_3) = \bigsqcup \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2 \cdot \ell_3\}$$

Now we obtain:

$$\ell_1 \cdot \ell \sqsubseteq \ell_2 \Rightarrow \ell_1 \cdot \ell \cdot \ell_3 \sqsubseteq \ell_2 \cdot \ell_3$$

and therefore $\{\ell \cdot \ell_3 \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \subseteq \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2 \cdot \ell_3\}$.

(v) Obviously, $\{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \subseteq \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_3\}$ and therefore:

$$\ell_1 \rightarrow \ell_2 = \bigsqcup \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \sqsubseteq \bigsqcup \{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_3\} = \ell_1 \rightarrow \ell_3$$

- (vi) Whenever $\ell_3 \sqsubseteq \ell_1 \rightarrow \ell_2$, then $\ell_1 \cdot \ell_3 \sqsubseteq \ell_1 \cdot (\ell_1 \rightarrow \ell_2) = \ell_1 \cdot \bigsqcup\{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} = \bigsqcup\{\ell_1 \cdot \ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \sqsubseteq \ell_2$.
Whenever $\ell_1 \cdot \ell_3 \sqsubseteq \ell_2$ we have that $\ell_1 \rightarrow \ell_2 = \bigsqcup\{\ell \in \mathbb{L} \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\} \supseteq \ell_3$, since ℓ_3 is an element of the set.
- (vii) Because of Lemma A.2.(vi) it suffices to show that $\ell_1 \cdot ((\ell_1 \rightarrow \ell_2) \sqcup (\ell_1 \rightarrow \ell_3)) = \ell_1 \cdot (\ell_1 \rightarrow \ell_2) \sqcup \ell_1 \cdot (\ell_1 \rightarrow \ell_3) \sqsubseteq \ell_2 \sqcup \ell_3$.
- (viii) $\ell_1 \cdot (\ell_1 \rightarrow \ell_2) = \ell_1 \cdot \bigsqcup\{\ell' \mid \ell_1 \cdot \ell' \sqsubseteq \ell_2\} = \bigsqcup\{\ell_1 \cdot \ell' \mid \ell_1 \cdot \ell' \sqsubseteq \ell_2\} \sqsubseteq \bigsqcup\{\ell_2\} = \ell_2$ \square

A.2 Congruence Closure

Congruence Closure for Rings

Lemma A.3.

- (i) Let \mathbb{L} be a ring. Let $R \subseteq \mathbb{L}^X \times \mathbb{L}^X$ be a congruence. Then $u(R)$ is a module.
- (ii) Let \mathbb{L} be a ring. Let $U \subseteq \mathbb{L}^X$ be a module. Then $r(U)$ is a congruence.

Proof.

- (i) We will show that $u(R)$ contains all vectors generated via linear combination from $u(R)$, making $u(R)$ a generating set for itself, i.e. a module.
- Let $v'' \in u(R)$ then there must be $v, v' \in \mathbb{L}^X$ such that $(v, v') \in R$ and $v - v' = v''$. Since R is congruence, it follows that $(v \cdot s, v' \cdot s) \in R$ for any $s \in \mathbb{L}$. This means that $v \cdot s - v' \cdot s \in u(R)$, distributivity now proves $(v - v') \cdot s \in u(R)$, i.e. $v'' \cdot s \in u(R)$.
 - Let $v_1'', v_2'' \in u(R)$. Then there must be $v_1, v_1', v_2, v_2' \in \mathbb{L}^X$ such that $v_i'' = v_i - v_i'$ and $(v_i, v_i') \in R$ for $i = 1, 2$. Since R is a congruence, it follows that $(v_1 + v_2, v_1' + v_2') \in R$. Thus, $(v_1 + v_2) - (v_1' + v_2') \in u(R)$. Commutativity of addition yields $(v_1 - v_1') + (v_2 - v_2') \in u(R)$, i.e. $v_1'' + v_2'' \in u(R)$.
- (ii) – *Reflexivity:* Let any $v \in U$ be given, then $v \cdot 0 \in U$, so the 0-vector is in U . For any given $v \in \mathbb{L}^X$, $v - v = 0$, thus $(v, v) \in r(U)$.
- *Symmetry:* Let $(v, v') \in r(U)$, then $v - v' \in U$. Since U is a module, $(v - v') \cdot (-1) \in U$ and thus $-v + v' = v' - v \in U$, therefore $(v', v) \in r(U)$.
 - *Transitivity:* Let $(v, v') \in r(U)$, $(v', v'') \in r(U)$, then $v - v' \in U$ and $v' - v'' \in U$. Since U is a module, $(v - v') + (v' - v'') \in U$ and thus $v - v'' \in U$. Therefore $(v, v'') \in r(U)$.
 - *Addition:* Let $(v_1, v_2) \in r(U)$ and $(v_1', v_2') \in r(U)$, then $v_1 - v_2 \in U$ and $v_1' - v_2' \in U$. Therefore $(v_1 - v_2) + (v_1' - v_2') \in U$. Commutativity yields $(v_1 + v_1') - (v_2 + v_2') \in U$, i.e. $(v_1 + v_1', v_2 + v_2') \in r(U)$.
 - *Multiplication:* Let $(v, v') \in r(U)$ and $s \in \mathbb{L}$. Then $v - v' \in U$. Since U is a module, $(v - v') \cdot s \in U$, distributivity yields $v \cdot s - v' \cdot s \in U$ and thus per definition $(v \cdot s, v' \cdot s) \in r(U)$.

\square

Proof of Proposition 3.1

Proof. We first define the following two functions:

- $u : \mathcal{P}(\mathbb{I}^X \times \mathbb{I}^X) \rightarrow \mathcal{P}(\mathbb{I}^X)$ with $u(R) = \{v - v' \mid (v, v') \in R\}$.
- $r : \mathcal{P}(\mathbb{I}^X) \rightarrow \mathcal{P}(\mathbf{R}^X \times \mathbb{I}^X)$ with $r(U) = \{(v, v') \mid v - v' \in U\}$, where $U \subseteq \mathbb{I}^X$, i.e., U is a set of \mathbb{I} -vectors.

According to Lemma A.3 in Appendix A we know that if R is a congruence, then $u(R)$ is a submodule and if U is a submodule then $r(U)$ is a congruence.

Observe that $R \subseteq R'$ implies $u(R) \subseteq u(R')$ via definition of u and $r(U) \subseteq r(U')$ whenever $U \subseteq U'$, by definition of r .

Observe furthermore that $r(u(R)) \subseteq c(R)$ holds, because if $(v_1, v_2) \in r(u(R))$, then there exists a $(v'_1, v'_2) \in R$ such that $v_1 - v_2 = v'_1 - v'_2$, hence $v'_1 - v_1 = v'_2 - v_2$. Now $(v'_1 - v_1, v'_1 - v_1) \in c(R)$ due to reflexivity and thus we obtain: $(v_1, v_2) + (v'_1 - v_1, v'_1 - v_1) = (v_1, v_2) + (v'_1 - v_1, v'_2 - v_2) = (v_1 + v'_1 - v_1, v_2 + v'_2 - v_2) = (v'_1, v'_2)$, hence $(v'_1, v'_2) \in c(R)$. Thus, $r(u(R)) \subseteq c(R)$, proving also that congruences are fixed points of the monotone function $r \circ u$, since $r(U)$ is always a congruence and for every congruence R it holds that $c(R) = R$.

Now we can observe that the module generated by a set of vectors is the smallest module that contains this set and the congruence closure of a relation is the smallest congruence closed relation containing that relation.

We will now show $r([U_R]) = r([u(R)]) = c(R)$, thus proving the statement of the proposition. We have $u(R) \subseteq u(c(R))$ and we know that $u(c(R))$ is a submodule from Lemma A.3.(i), hence the submodule generated by $u(R)$ is included in $u(c(R))$, i.e. $[u(R)] \subseteq u(c(R))$. Therefore, $r([u(R)]) \subseteq r(u(c(R))) = c(R)$, and since Lemma A.3.(ii) shows that r applied to a submodule yields a congruence and we have $R \subseteq r([u(R)]) \subseteq c(R)$, the second inclusion is indeed an equality. \square

Congruence Closure for l -Monoids

Proof of Lemma 3.4

Proof.

- (i) Assume that $v \rightsquigarrow v'$, via rule $l \mapsto r$, and $v \sqsubseteq w$. Then $v' = v \sqcup r \cdot (l \rightarrow v) \sqsubseteq w \sqcup r \cdot (l \rightarrow w) =: w'$. Hence either $w \rightsquigarrow w'$ or $w = w'$ and in this case $w \sqsupseteq v'$.
- (ii) Assume that $v \rightsquigarrow v'$, via rule $l \mapsto r$. Define $u := (v \sqcup w) \sqcup r \cdot (l \rightarrow (v \sqcup w)) \sqsupseteq (v \sqcup w) \sqcup r \cdot ((l \rightarrow v) \sqcup (l \rightarrow w)) = (v \sqcup r \cdot (l \rightarrow v)) \sqcup (w \sqcup r \cdot (l \rightarrow w)) \sqsupseteq v' \sqcup w$. The first inequality is due to Lemma A.2.(vii).
Now either $v \sqcup w \rightsquigarrow u$ or $v \sqcup w = u \sqsupseteq v' \sqcup w$. Since we also have that $v \sqcup w \sqsubseteq v' \sqcup w$, this implies $v \sqcup w = v' \sqcup w$.

\square

Proof of Lemma 3.6

Proof. Assume that $v \rightsquigarrow v_1$ and $v \rightsquigarrow v_2$. We set $v_0^a = v$, $v_1^a = v_1$ and consider a sequence of rewriting steps $v_1^a \rightsquigarrow v_2^a \rightsquigarrow \dots \rightsquigarrow v_n^a \not\rightsquigarrow$ that leads to normal form v_n^a .

We now construct a sequence of vectors v_1^b, \dots, v_{n+1}^b where $v_1^b = v_2$, $v_i^b \rightsquigarrow v_{i+1}^b$ or $v_i^b = v_{i+1}^b$, and $v_{i+1}^b \sqsupseteq v_i^a$.

Given v_i^b with $i \geq 1$, Lemma 3.4.(i) guarantees the existence of $v_{i+1}^b \sqsupseteq v_i^a$ with $v_i^b \rightsquigarrow v_{i+1}^b$, or $v_i^b \sqsupseteq v_i^a$. In the latter case we set $v_{i+1}^b = v_i^b$.

Since $v \rightsquigarrow^* v_{n+1}^b$ and v_n^a is a normal form, it must hold that $v_{n+1}^b \leq v_n^a$. We also know from above that $v_{n+1}^b \sqsupseteq v_n^a$, hence $v_{n+1}^b = v_n^a$. Hence, this is the vector v' which is reachable from both v_1 and v_2 and which proves the local Church-Rosser property. \square

Proof of Lemma 3.7

Proof. Assume that $v \rightsquigarrow v'$ via rule $l \mapsto r$. Hence $v' = v \sqcup r \cdot (l \rightarrow v)$. Let $v'' := v \cdot \ell \sqcup r \cdot (l \rightarrow v \cdot \ell)$. By adapting the proof of Lemma A.2.(iv) to vectors we can show that $v'' \sqsupseteq v \cdot \ell \sqcup r \cdot (l \rightarrow v) \cdot \ell = (v \sqcup r \cdot (l \rightarrow v)) \cdot \ell = v' \cdot \ell$. Hence either $v \cdot \ell \rightsquigarrow v''$ or $v \cdot \ell = v''$. In the latter case $v \cdot \ell \sqsupseteq v' \cdot \ell$ (since $v'' \sqsupseteq v' \cdot \ell$), but also $v \cdot \ell \sqsubseteq v' \cdot \ell$ (since $v \sqsubseteq v'$). Hence $v \cdot \ell = v' \cdot \ell$.

Now assume that $v = v_0 \rightsquigarrow v' = v_1 \rightsquigarrow v_2 \rightsquigarrow \dots \rightsquigarrow v_n = \Downarrow v$. We construct a sequence of vectors w_i where $w_0 = v \cdot \ell$, $w_i \sqsupseteq v_i \cdot \ell$ and either $w_i \rightsquigarrow w_{i+1}$ or $w_i = w_{i+1}$.

Given $w_i \sqsupseteq v_i \cdot \ell$ and $v_i \rightsquigarrow v_{i+1}$. We know that one of the following two cases holds:

- there exists v'' such that $v_i \cdot \ell \rightsquigarrow v'' \sqsupseteq v_{i+1} \cdot \ell$: now, since $w_i \sqsupseteq v_i \cdot \ell$, we know due to Lemma 3.4.(i) that there exists w_{i+1} such that $w_i \rightsquigarrow w_{i+1} \sqsupseteq v'' \sqsupseteq v_{i+1} \cdot \ell$ or $w_i \sqsupseteq v''$. In the second case we set $w_{i+1} = w_i$.
- or $v_i \cdot \ell = v_{i+1} \cdot \ell$: again we set $w_{i+1} = w_i$ and obtain $w_{i+1} = w_i \sqsupseteq v_i \cdot \ell = v_{i+1} \cdot \ell$.

Hence $w_n \sqsupseteq v_n = \Downarrow v$ and via monotonicity we obtain $\Downarrow (v \cdot \ell) = \Downarrow w_0 = \Downarrow w_n \sqsupseteq \Downarrow v_n = \Downarrow v$. \square

Proof of Theorem 3.8

Proof. It suffices to show that $\Downarrow v \sqsupseteq v'$ and $\Downarrow v' \sqsupseteq v$, because if $v \sqsubseteq \Downarrow v'$, then $\Downarrow v \sqsubseteq \Downarrow v'$ (\Downarrow is monotone and idempotent) and vice-versa. We prove this via rule induction (cf. rules in Table 1).

(REL) If we find that $v \text{ c}(R) v'$ because $v R v'$, then there are rules $v \mapsto v \sqcup v'$ and $v' \mapsto v \sqcup v'$.

Hence v rewrites to $v \sqcup (v \sqcup v') \cdot (v \rightarrow v) \geq v \sqcup v'$, since $v \rightarrow v \sqsupseteq 1$ (or v can not be rewritten via this rule). Hence either v is rewritten to a vector larger or equal v' or $v \sqsupseteq v'$ holds. Hence $\Downarrow v \sqsupseteq v'$.

Analogously one can show $\Downarrow v' \sqsupseteq v$.

- (REFL) If we find that $v \ c(R) \ v'$ because of reflexivity (i.e. $v = v'$), then trivially $\Downarrow v = \Downarrow v'$
- (SYM) If we find that $v \ c(R) \ v'$ because of symmetry, then we already know from the induction hypothesis that $\Downarrow v \sqsupseteq v'$ and $\Downarrow v' \sqsupseteq v$.
- (TRANS) If we find that $v_1 \ c(R) \ v_3$ because of transitivity, i.e. $v_1 \ c(R) \ v_2$ and $v_2 \ c(R) \ v_3$, we know from the induction hypothesis that $\Downarrow v_1 \sqsupseteq v_2$ and $\Downarrow v_2 \sqsupseteq v_3$ as well as $\Downarrow v_3 \sqsupseteq v_2$ and $\Downarrow v_2 \sqsupseteq v_1$. In particular, we have $v_1 \leq \Downarrow v_2 \sqsubseteq \Downarrow v_3$ and $v_3 \leq \Downarrow v_2 \sqsubseteq \Downarrow v_1$.
- (SCA) If we find that $v \cdot \ell \ c(R) \ v' \cdot \ell$ because $v \ c(R) \ v'$, then $v \sqsubseteq \Downarrow v'$ and therefore, using Lemma 3.7

$$v \cdot \ell \sqsubseteq (\Downarrow v) \cdot \ell \sqsubseteq (\Downarrow v') \cdot \ell \sqsubseteq \Downarrow (v' \cdot \ell)$$

- (PLUS) If we find that $\bar{v} \sqcup v \ c(R) \ \bar{v}' \sqcup v'$ because of $\bar{v} \ c(R) \ \bar{v}'$ and $v \ c(R) \ v'$, then

$$v \sqcup \bar{v} \sqsubseteq (\Downarrow v') \sqcup (\Downarrow \bar{v}') \sqsubseteq \Downarrow (v' \sqcup \bar{v}'),$$

due to the monotonicity of \Downarrow .

□

Proof of Proposition 3.9

Proof. We prove this via rule induction (cf. rules in Table 1).

- (REL) If we find that $v \ c(R) \ \bar{v}$ because $v \ R \ \bar{v}$, then there are rules $v \mapsto v \sqcup \bar{v}$ and $\bar{v} \mapsto v \sqcup \bar{v}$. As in the proof of Theorem 3.8 we obtain that v rewrites to a vector larger or equal $v \sqcup \bar{v}$ in one step or that v itself has this property. Analogously for \bar{v} .
- (REFL) If we find that $v \ c(R) \ \bar{v}$ because of reflexivity (i.e. $\bar{v} = v$), then no rewriting step is needed.
- (SYM) If we find that $v \ c(R) \ \bar{v}$ because of symmetry, then we already know this from the induction hypothesis because the property we want to prove is symmetric.
- (TRANS) If we find that $v_1 \ c(R) \ v_3$ because of transitivity, i.e. $v_1 \ c(R) \ v_2$ and $v_2 \ c(R) \ v_3$, we know inductively: $v_1 \rightsquigarrow^* v'_1 \sqsupseteq v_1 \sqcup v_2$, $v_2 \rightsquigarrow^* v'_2 \sqsupseteq v_2 \sqcup v_3$. Now, due to Lemma 3.4.(ii) $v_1 \sqcup v_2 \rightsquigarrow^* u \sqsupseteq v_1 \sqcup v'_2 \sqsupseteq v_1 \sqcup v_2 \sqcup v_3 \sqsupseteq v_1 \sqcup v_3$. Furthermore since $v'_1 \sqsupseteq v_1 \sqcup v_2$ we know from Lemma 3.4.(i) that $v_1 \rightsquigarrow^* v'_1 \rightsquigarrow^* v''_1 \sqsupseteq u$. Combined, we obtain $v_1 \rightsquigarrow^* v''_1 \sqsupseteq v_1 \sqcup v_3$. For v_3 the proof is analogous.
- (SCA) If we have $v_1 \cdot \ell \ c(R) \ v_2 \cdot \ell$ because $v_1 \ c(R) \ v_2$ then $v_1 \rightsquigarrow v'_1 \sqsupseteq v_1 \sqcup v_2$ and $v_2 \rightsquigarrow v'_2 \sqsupseteq v_1 \sqcup v_2$. Thus, using Lemma 3.7

$$v_1 \cdot \ell \rightsquigarrow^* v''_1 \sqsupseteq v'_1 \cdot \ell \sqsupseteq (v_1 \sqcup v_2) \cdot \ell = v_1 \cdot \ell \sqcup v'_1 \cdot \ell$$

$$v_2 \cdot \ell \rightsquigarrow^* v''_2 \sqsupseteq v'_2 \cdot \ell \sqsupseteq (v_1 \sqcup v_2) \cdot \ell = v_1 \cdot \ell \sqcup v'_1 \cdot \ell$$

- (PLUS) If we have $v_1 \sqcup v_2 \ c(R) \ v'_1 \sqcup v'_2$ because of $v_1 \ c(R) \ v'_1$ and $v_2 \ c(R) \ v'_2$, then $v_1 \rightsquigarrow^* v''_1 \sqsupseteq v_1 \sqcup v'_1$ and $v_2 \rightsquigarrow^* v''_2 \sqsupseteq v_2 \sqcup v'_2$ and we obtain with Lemma 3.4:

$$v_1 \sqcup v_2 \rightsquigarrow^* v \sqsupseteq v''_1 \sqcup v''_2 \sqsupseteq (v_1 \sqcup v'_1) \sqcup (v_2 \sqcup v'_2) = (v_1 \sqcup v_2) \sqcup (v'_1 \sqcup v'_2).$$

Analogously for v'_1, v'_2 .

□

Proof of Corollary 3.10

Proof. Take $\bar{v} = \bigsqcup \{\hat{v} \mid v \rightsquigarrow^* \hat{v}\}$. By Proposition 3.9 if $v \in c(R)$ \bar{v} , then $v \rightsquigarrow^* \bar{v}$. Since \rightsquigarrow is irreflexive, $\bar{v} \not\rightsquigarrow$ and \bar{v} is in normal form.

If we assume that each rule that is applicable is applied at one point (or rendered unapplicable by other rule applications), it is sufficient to know that there exists one rewriting sequence reaching \bar{v} from v . If we decide to apply a rule, that was applied in this specific sequence, at a later point of time, either we have already exceeded the corresponding vector in the original rewriting sequence via other rule applications, or the rule can still be applied with the same or a greater multiplicand, leading again to a larger vector.

Hence every sequence of rewriting steps will eventually reach \bar{v} . \square

Termination for Specific l -Monoids

Proof of Theorem 3.11

Proof. We show this via contradiction. So we assume the algorithm does not terminate, i.e. there exists an infinite sequence of rewriting steps starting from a vector v . If that is the case, observe that there are some indices such that the corresponding entries in the vector increase infinitely often. We can assume that from the beginning, there are only indices that increase infinitely often or do not increase at all, because otherwise, we can apply rules until this is true and use the resulting vector as the new starting vector. Equivalently we can assume that each rule is applied infinitely often, by applying rules until no rule that can only be applied finitely often can ever get applied anymore and then removing all these rules from the rule system.

We call the initial vector v and the rule system \mathcal{R} . The sequence of intermediate rewriting results is a sequence v_0, v_1, \dots where $v_i \rightsquigarrow v_{i+1}$ for all $i \in \mathbb{N}_0$ in a single rewriting step. Taking a look at the history of a specific component $v[j]$ of v , we can observe that in each rewriting step applying a rule $l \mapsto r$, $v[j]$ either does not change or is rewritten to $\frac{r[j] \cdot v[j']}{l[j']}$ for a vector-index j' . In fact, we choose the index j' which minimizes that quotient. Inductively, we obtain that at any given rewriting step i ,

$$v_i[j] = \frac{r_n[j] \cdot r_{n-1}[j_{n-1}] \cdot \dots \cdot r_1[j_1] \cdot v[j_0]}{l_n[j_{n-1}] \cdot l_{n-1}[j_{n-2}] \cdot \dots \cdot l_1[j_0]}$$

where j_0, \dots, j_n are vector-indices and $l_1 \mapsto r_1, \dots, l_n \mapsto r_n$ are rules. The maximum index of rules is not the same as i , because only those rules are multiplied that really contributed to the value of $v_i[j]$ directly, thus, n might be smaller than i . Note that we used the fact that multiplication with 1 in order to apply a rule cannot happen, since then the rule could never be applied again. (In this case we say that the rule has been applied maximally.) If a rule was used maximally instead, it would not necessarily contribute a factor of the type $\frac{r[j]}{l[j]}$. Note that this representation is unique and we say $v_i[j]$ is based on $v[j_0]$ if it can be written as above.

Let N be the dimension of v . At any given time i , there are at most N different entries in v_i and each entry in v_{i+1} is obtained by multiplying one factor of the form

$\frac{r[\bar{i}]}{l[\bar{j}]}$, where $l \mapsto r$ is a rule and \bar{i}, \bar{j} are vector-indices, with one of the entries in v_i , or is identical to the entry in v_i . After at most $N \cdot (N - 1) + 1$ steps, there must exist one vector-index j , such that there exist indices $i \leq i' < j' \leq i + N \cdot (N - 1) + 1$ where $v_{i'}[j]$ and $v_{j'}[j]$ are not identical and based on the same entry $\ell \in [0, 1]$ from v_i , i.e. $v_{i'}[j]$ can be written as

$$v_{i'}[j] = \frac{r_k[j] \cdot r_{k-1}[i_{k-1}] \cdot \dots \cdot r_1[i_1]}{l_k[i_{k-1}] \cdot \dots \cdot l_2[i_1] \cdot l_1[i_0]} \cdot \ell$$

and $v_{j'}[j]$ as

$$v_{j'}[j] = \frac{r'_h[j] \cdot r'_{h-1}[j_{h-1}] \cdot \dots \cdot r'_1[j_1]}{l_h[j_{h-1}] \cdot \dots \cdot l'_2[j_1] \cdot l'_1[j_0]} \cdot \ell$$

where k and h are at most $N \cdot (N - 1)$.

This can be proven as follows: Each vector v_i has at most N different entries, so there are at most N different entries from v_i an entry in a vector v_j , $j \geq i$, can be based on. In each step, at least one entry of v_i is rewritten to something larger. Each of the N entries of v_i can be rewritten and increased in the process at most $N - 1$ times, without being based on the same element from v_i twice (including the initial entry of v_i), due to the pigeonhole principle. Again, since in each rewriting step at least one entry gets changed and there are only N entries, after $N \cdot (N - 1) + 1$ rewriting steps, there must be at least one entry that was based on the same entry from v_i twice and increased inbetween.

Thus, we have

$$v_{j'}[j] = \frac{r'_h[j] \cdot r'_{h-1}[j_{h-1}] \cdot \dots \cdot r'_1[j_1]}{l_h[j_{h-1}] \cdot \dots \cdot l'_2[j_1] \cdot l'_1[j_0]} \cdot \frac{l_k[i_{k-1}] \cdot \dots \cdot l_2[i_1] \cdot l_1[i_0]}{r_k[j] \cdot r_{k-1}[i_{k-1}] \cdot \dots \cdot r_1[i_1]} \cdot v_{i'}[j]$$

which means, $v_{j'}[j]$ can be obtained from $v_{i'}[j]$ via multiplication of at most $N \cdot (N - 1) + 1$ factors of the form $\frac{r[\bar{i}]}{l[\bar{j}]}$ and at most $N \cdot (N - 1) + 1$ factors of the form $\frac{l[\bar{i}]}{r[\bar{j}]}$. Also, since $v_{j'}[j] > v_{i'}[j]$, this multiplicand is larger than 1. Observe that due to finiteness of \mathcal{R} , there are only finitely many products of at most $N \cdot (N - 1) + 1$ factors of the form $\frac{r[\bar{i}]}{l[\bar{j}]}$ and at most $N \cdot (N - 1) + 1$ factors of the form $\frac{l[\bar{i}]}{r[\bar{j}]}$, so there is a least one such factor $\delta > 1$. Moreover, this construction works for each interval of size $N \cdot (N - 1)$. Dividing the whole history of rule applications into consecutive chunks of size $N \cdot (N - 1) + 1$ yields infinitely many intervals where in each interval at least one index increases by at least factor δ . Since the dimension of v is finite, there must be at least one index j which has this property infinitely often. That means that $v[j]$ is rewritten to something larger than $\delta^n \cdot v[j]$ for each $n \in \mathbb{N}_0$. However, the sequence $\langle \delta^n \rangle$ is not bounded, therefore $\delta^n \cdot v[j]$ is not bounded by 1, but that is a contradiction to the assumption that the rewriting never terminates.

Using this result, we can now go on to show that rewriting terminates for the tropical semiring as well.

We show this by proving that the tropical semiring $(\mathbb{R}_0^+ \cup \{\infty\}, \min, +\infty, 0)$ and $([0, 1], \max, \cdot, 0, 1)$ are isomorphic with an isomorphism that is compatible with the

order and the multiplication, which ensures that any given transformation system in one of those semirings can do a transformation step iff the transformation system obtained by applying the isomorphism to each component of every rule as well as the vector under consideration can do one.

We use the bijection $f : \mathbb{R}_0^+ \cup \{\infty\} \rightarrow [0, 1]$, $f(x) = 2^{-x}$ where we have extended the power to $-\infty$ via the natural definition $2^{-\infty} = 0$. Obviously, this function is bijective with the inverse being $f^{-1}(x) = -\log_2(x)$, where the base-2 logarithm is extended to 0 via $\log_2(0) = -\infty$. Hence we only have to prove that f respects the order of the lattices and that it is compatible with addition and multiplication.

– Let $\ell_1, \ell_2 \in \mathbb{R}_0^+ \cup \{\infty\}$ be given, then

$$\ell_1 \geq \ell_2 \Leftrightarrow -\ell_1 \leq -\ell_2 \Leftrightarrow 2^{-\ell_1} \leq 2^{-\ell_2} \Leftrightarrow f(\ell_1) \leq f(\ell_2).$$

Note that the order is swapped between the two semirings, so the function indeed is an order-isomorphism.

– Let $\ell_1, \ell_2 \in \mathbb{R}_0^+ \cup \{\infty\}$ be given, then

$$f(\min\{\ell_1, \ell_2\}) = 2^{-\min\{\ell_1, \ell_2\}} = \max\{2^{-\ell_1}, 2^{-\ell_2}\} = \max\{f(\ell_1), f(\ell_2)\}$$

– Let $\ell_1, \ell_2 \in \mathbb{R}_0^+ \cup \{\infty\}$ be given, then

$$f(\ell_1 + \ell_2) = 2^{-(\ell_1 + \ell_2)} = 2^{-\ell_1} \cdot 2^{-\ell_2} = f(\ell_1) \cdot f(\ell_2)$$

□

Termination for Lattices

Lemma A.4. *Let $\ell_1, \ell_2 \in \mathbb{L}$, where \mathbb{L} is a lattice, then*

- (i) $\ell_1 \rightarrow_{\mathbb{L}} \ell_2 \sqsupseteq \ell_2$
- (ii) $\ell_1 \rightarrow_{\mathbb{L}} \ell_2 \sqsupseteq \lfloor \neg \ell_1 \rfloor$
- (iii) $\ell_1 \rightarrow_{\mathbb{L}} \ell_2 = \lfloor \ell_1 \rightarrow_{\mathbb{B}} \ell_2 \rfloor$
- (iv) $\lfloor \neg \ell_1 \rfloor \sqcup_{\mathbb{B}} \ell_2 \sqsubseteq \ell_1 \rightarrow_{\mathbb{L}} \ell_2 \sqsubseteq \neg \ell_1 \sqcup_{\mathbb{B}} \ell_2$
- (v) $\ell_1 \rightarrow_{\mathbb{L}} \ell_2$ can be written as $\ell_1^* \sqcup_{\mathbb{B}} \ell_2$ for an $\lfloor \neg \ell_1 \rfloor \sqsubseteq \ell_1^* \sqsubseteq \neg \ell_1$.

Proof.

- (i) Every lattice is integral, hence $\ell_1 \cdot \ell_2 \sqsubseteq \ell_2$. Hence ℓ_2 is an element of the set $\{\ell \mid \ell_1 \cdot \ell \sqsubseteq \ell_2\}$, the supremum of which is $\ell_1 \rightarrow_{\mathbb{L}} \ell_2$. Hence $\ell_1 \rightarrow_{\mathbb{L}} \ell_2 \geq \ell_2$.
- (ii) Per definition, $\lfloor \neg \ell_1 \rfloor \cap \ell_1 \sqsubseteq \neg \ell_1 \cap \ell_2 = \perp_{\mathbb{B}}$ and if there were an element $\ell \sqsupset \perp_{\mathbb{L}}$ such that $\lfloor \neg \ell_1 \rfloor \sqsupseteq \ell$ and $\ell_1 \sqsupseteq \ell$, then this would be true in \mathbb{B} , too, and therefore such a ℓ cannot exist. Thus, in particular, $\lfloor \neg \ell_1 \rfloor \cap \ell_1 = \perp_{\mathbb{L}} \sqsubseteq \ell_2$ and per definition of $\ell_1 \rightarrow_{\mathbb{L}} \ell_2$, this proves that $\lfloor \neg \ell_1 \rfloor \sqsubseteq \ell_1 \rightarrow_{\mathbb{L}} \ell_2$.
- (iii) We observe that

$$\lfloor \ell_1 \rightarrow_{\mathbb{B}} \ell_2 \rfloor = \bigsqcup_{\mathbb{L}} \{\ell \in \mathbb{L} \mid \ell \sqsubseteq \ell_1 \rightarrow_{\mathbb{B}} \ell_2\}$$

and

$$\ell_1 \rightarrow_{\mathbb{L}} \ell_2 = \bigsqcup_{\mathbb{L}} \{\ell \in \mathbb{L} \mid \ell_1 \sqcap \ell \sqsubseteq \ell_2\}$$

We will show that both sets are equal:

– \subseteq :

$$\ell \sqsubseteq (\ell_1 \rightarrow_{\mathbb{B}} \ell_2) \Rightarrow \ell_1 \sqcap \ell \sqsubseteq \ell_1 \sqcap (\ell_1 \rightarrow_{\mathbb{B}} \ell_2)$$

But then:

$$\ell_1 \sqcap \ell \sqsubseteq \ell_1 \sqcap (\ell_1 \rightarrow_{\mathbb{B}} \ell_2) \sqsubseteq \ell_2$$

And thus $\ell \in \{\ell' \in \mathbb{L} \mid \ell_1 \sqcap \ell' \sqsubseteq \ell_2\}$.

– \supseteq :

$$\ell_1 \sqcap \ell \sqsubseteq \ell_2 \Rightarrow \ell \in \{\ell' \in \mathbb{L} \mid \ell_1 \sqcap \ell' \sqsubseteq \ell_2\} \subseteq \{\ell' \in \mathbb{B} \mid \ell_1 \sqcap \ell' \sqsubseteq \ell_2\}$$

Hence ℓ is smaller or equal than the supremum of this set.

$$(iv) \quad \lfloor \neg \ell_1 \rfloor \sqcup_{\mathbb{B}} \ell_2 \sqsubseteq \ell_1 \rightarrow_{\mathbb{L}} \ell_2 = \lfloor \ell_1 \rightarrow_{\mathbb{B}} \ell_2 \rfloor = \lfloor \neg \ell_1 \sqcup_{\mathbb{B}} \ell_2 \rfloor \sqsubseteq \neg \ell_1 \sqcup_{\mathbb{B}} \ell_2$$

Lemma A.4.(i) · Lemma A.4.(ii) Lemma A.4.(iii)

(v) In this proof we are exclusively computing in \mathbb{B} , so we do not point out that $\sqcup = \sqcup_{\mathbb{B}}$. We will define $\ell_1^* := ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2) \sqcup \lfloor \neg \ell_1 \rfloor$ and first prove that $\ell_1 \rightarrow_{\mathbb{L}} \ell_2 = \ell_1^* \sqcup \ell_2$ and then prove that $\lfloor \neg \ell_1 \rfloor \sqsubseteq \ell_1^* \sqsubseteq \neg \ell_1$.

For the first part of the proof:

$$\begin{aligned} \ell_1 \rightarrow_{\mathbb{L}} \ell_2 &= \ell_1 \rightarrow_{\mathbb{L}} \ell_2 \sqcup \lfloor \neg \ell_1 \rfloor \sqcup \ell_2 \\ &= ((\ell_2 \sqcup \neg \ell_2) \sqcap (\ell_1 \rightarrow_{\mathbb{L}} \ell_2)) \sqcup \lfloor \neg \ell_1 \rfloor \sqcup \ell_2 \\ &= ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2) \sqcup ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \ell_2) \sqcup \lfloor \neg \ell_1 \rfloor \sqcup \ell_2 \\ &= ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2) \sqcup \lfloor \neg \ell_1 \rfloor \sqcup \ell_2 \\ &= \ell_1^* \sqcup \ell_2 \end{aligned}$$

Lemma A.4.(iv)

Now, obviously, $\lfloor \neg \ell_1 \rfloor \sqsubseteq ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2) \sqcup \lfloor \neg \ell_1 \rfloor = \ell_1^*$, since $\lfloor \neg \ell_1 \rfloor$ is part of the supremum. It is only left to be shown that $((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2) \sqcup \lfloor \neg \ell_1 \rfloor \sqsubseteq \neg \ell_1$ holds as well. First, we observe, that $\lfloor \neg \ell_1 \rfloor \sqsubseteq \neg \ell_1$ per definition, so it suffices to show that $(\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2 \sqsubseteq \neg \ell_1$. Moreover, it is true that $\neg \ell_1 = \bigsqcup \{\ell \in \mathbb{B} \mid \ell \sqcap \ell_1 = \perp\}$. Then a simple computation shows:

$$(\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2 \sqcap \ell_1 = ((\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \ell_1) \sqcap \neg \ell_2 \sqsubseteq \ell_2 \sqcap \neg \ell_2 = \perp.$$

Thus,

$$(\ell_1 \rightarrow_{\mathbb{L}} \ell_2) \sqcap \neg \ell_2 \in \{\ell \in \mathbb{B} \mid \ell \sqcap \ell_1 = \perp\}$$

of which $\neg \ell_1$ is the supremum.

□

Proof of Theorem 3.12

Proof. Lemma A.4.(v) shows that each multiplicand $l \rightarrow_{\mathbb{L}} v$ can be written as supremum of $v[x]$ for an index $x \in X$ and an element l^* from the finite set $L(l, x)$. This set is independent of v , the element from the set must however be chosen according to v . Therefore, each element we obtain in rewriting is built as infimum and supremum of finitely many elements from \mathbb{B} and – using conjunctive normal form – we obtain that we can only build finitely many different rewriting results from v . Therefore, rewriting terminates for every vector v . \square

A.3 Up-To Techniques for Weighted Automata

Coinduction and Up-to Techniques

The soundness of the algorithms in Section 4 can be proved in a clear way by exploiting coinduction and up-to techniques. In this appendix we shortly recall the essential results of the theory developed in [20]. We fix the lattice of relations over \mathbb{S}^X , $Rel_{\mathbb{S}^X}$, but the results expressed here hold for arbitrary complete lattices.

Given a monotone map $b: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$, the Knaster-Tarski fixed-point theorem characterises the greatest fixed-point νb as the union of all post-fixed points of b :

$$\nu b = \bigcup \{R \subseteq \mathbb{S}^X \times \mathbb{S}^X \mid R \subseteq b(R)\}.$$

This immediately leads to the *coinduction proof principle*

$$\frac{\exists R, S \subseteq R \subseteq b(R)}{S \subseteq \nu b} \quad (1)$$

which allows to prove $(v_1, v_2) \in \nu b$ by exhibiting a post-fixed-point R such that $\{(v_1, v_2)\} \subseteq R$. We call the post-fixed-points of b , *b-simulations*. For a monotone map $f: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$, a *b-simulation up-to f* is a relation R such that $R \subseteq b(f(R))$. We say that f is *compatible with b* if $f(b(R)) \subseteq b(f(R))$ for all relations R . The following result from [20] justifies our interest in compatible up-to techniques.

Theorem A.5. *If f is b-compatible and $R \subseteq b(f(R))$ then $R \subseteq \nu b$.*

The above theorem leads to the coinduction up-to principle

$$\frac{\exists R, S \subseteq R \subseteq b(f(R))}{S \subseteq \nu b} \quad (2)$$

Up-to techniques can be combined in a number of interesting ways. For a map $f: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$, the n -iteration of f is defined as $f^{n+1} = f \circ f^n$ and $f^0 = id$, the identity function. The omega iteration is defined as $f^\omega(R) = \bigcup_{i=0}^{\infty} f^i(R)$. Given two relations R and S , we use $R \bullet S$ to denote their relational composition $\{(x, z) \mid \exists y \mid (x, y) \in R \text{ and } (y, z) \in S\}$. For two functions, $f, g: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$, we write $f \bullet g$ for the function mapping a relation R into $f(R) \bullet g(R)$.

The following result from [20] informs us that compatible up-to techniques can be composed resulting in other compatible techniques.

Lemma A.6. *The following functions are b -compatible:*

- id : the identity function;
- $f \circ g$: the composition of b -compatible functions f and g ;
- $\bigcup F$: the pointwise union of an arbitrary family F of b -compatible functions: $\bigcup F(R) = \bigcup_{f \in F} f(R)$;
- f^ω : the (omega) iteration of a b -compatible function f , defined as $f^\omega(R) = \bigcup_{i=0}^\infty f^i(R)$

Moreover, if $b(R) \bullet b(S) \subseteq b(R \bullet S)$ for all relations R, S

- $f \bullet g$: the relation composition of b -compatible functions f and g ;

is b -compatible.

With these results it is easy to prove the soundness of the discussed algorithms.

Language Equivalence for Weighted Automata

Proof of Proposition 4.2

Proof. The key step consists in characterising \sim as νb_1 . This follows easily from abstract results (see e.g. [6,15]), but a concrete proof would proceed as follows:

1. prove that b_1 is a co-continuous function, i.e., $b_1(\bigcap_{i=1}^\infty S_i) = \bigcap_{i=1}^\infty b_1(S_i)$ whenever $S_{i+1} \subseteq S_i$;
2. therefore, by the Kleene fixed-point theorem [11], $\nu b_1 = \bigcap_n b_1^n(\top)$ where $b_1^0 = Id$ and $b_1^{n+1} = b_1 \circ b_1^n$;
3. prove, using induction, that for all n , $b_1^n(\top) = \{(v_1, v_2) \mid \llbracket v_1 \rrbracket(w) = \llbracket v_2 \rrbracket(w) \text{ for all words } w \in A^* \text{ up to length } n-1\}$;
4. conclude by 2 and 3 that $\nu b_1 = \{(v_1, v_2) \mid \llbracket v_1 \rrbracket = \llbracket v_2 \rrbracket \text{ for all word } w \in A^*\}$.

By coinduction, the first statement follows.

For the second statement we have to use coinduction up-to and prove b_1 -compatibility of c . The latter follows from abstract results [7]. For a concrete proof, one has first to show that the following monotone maps are b_1 -compatible.

- the constant reflexive function: $r(R) = \{(x, x) \mid x \in S\}$;
- the converse function: $s(R) = \{(y, x) \mid (x, y) \in R\}$;
- the squaring function: $t(R) = \{(x, z) \mid \exists y, (x, y) \in R \text{ and } (y, z) \in R\}$.
- the sum function: $+(R) = \{(v_1 + v_2, v'_1 + v'_2) \mid (v_1, v_2) \in R \text{ and } (v'_1, v'_2) \in R\}$.
- the scalar function: $\cdot(R) = \{(v \cdot s, w \cdot s) \mid (v, w) \in R \text{ and } s \in \mathbb{S}\}$.

Then, one observe that $c = (Id \cup r \cup s \cup t \cup + \cup \cdot)^\omega$ and conclude that c is b -compatible by Lemma A.6. \square

Language Inclusion for Weighted Automata

Lemma A.7. $\nu b_2 = \sqsubseteq$.

Proof. The proof proceeds as for the first part of Proposition 4.2 by using \sqsubseteq in place of $=$ and b_2 in place of b_1 . \square

Lemma A.8. *If \sqsubseteq is a precongruence, the following monotone maps are b_2 -compatible:*

- the constant ord function: $\sqsubseteq(R) = \{(v_1, v_2) \mid v_1 \sqsubseteq v_2\}$;
- the constant inclusion function: $\sqsubseteq(R) = \{(v_1, v_2) \mid v_1 \sqsubseteq v_2\}$;
- the squaring function: $t(R) = \{(v_1, v_3) \mid \exists v_2, (v_1, v_2) \in R \text{ and } (v_2, v_3) \in R\}$.
- the sum function: $+(R) = \{(v_1 + v_2, v'_1 + v'_2) \mid (v_1, v_2) \in R \text{ and } (v'_1, v'_2) \in R\}$.
- the scalar function: $\cdot(R) = \{(v \cdot s, w \cdot s) \mid (v, w) \in R \text{ and } s \in \mathbb{S}\}$.

Proof. Since \sqsubseteq is a precongruence, if $v_1 \sqsubseteq v_2$, then $o(v_1) \sqsubseteq o(v_2)$ and for all $a \in A$, $t_a(v_1) \sqsubseteq t_a(v_2)$. Which means that $\sqsubseteq \subseteq b_2(\sqsubseteq)$, that is, for all relations R , $\sqsubseteq(b_2(R)) \subseteq b_2(\sqsubseteq(R))$. This proves the first statement. The others are similar. \square

Lemma A.9. p is compatible.

Proof. Observe that by definition $p = (Id \cup \sqsubseteq \cup t \cup + \cup \cdot)^\omega$. The statement follows immediately by Lemma A.6 and Lemma A.8. \square

Proof of Theorem 4.4

Proof. For soundness, observe that the following is an invariant for the while loop at step (3).

$$R \subseteq b_2(p(R) \cup \text{todo}) \quad (3)$$

If HKP returns *true* then *todo* is empty and thus $R \subseteq b_2(p(R))$, i.e., R is a b_2 -simulation up-to p . By Lemma A.9, Theorem A.5 and Lemma A.7, $v_1 \sqsubseteq v_2$.

For completeness, we proceed in the same way as in Theorem 4.3. \square

Proof of Proposition 4.5

Proof. This proof is very close in structure to the proofs for Theorem 3.5 and Theorem 3.8. We will use Lemma 3.6 and Lemma 3.7 because the claims and proofs for these lemmas can be copied verbatim for the asymmetric case, so we do not prove these claims again. This does not hold true for Theorem 3.5 and Theorem 3.8, though, where symmetry is relevant. We will now prove the two claims, adjusted to the non-symmetric case.

- Whenever there exists a vector $v'_2 \geq v_1$ such that v_2 rewrites to v'_2 via \mathcal{R} , i.e., $v_2 \rightsquigarrow_{\mathcal{R}}^* v'_2$, then $(v_1, v_2) \in p(R)$. This is the analogue to Theorem 3.5.

We will show that if $v_2 \rightsquigarrow_{\mathcal{R}}^* v'_2$, then $(v'_2, v_2) \in p(R)$. Furthermore $v_1 \leq v'_2$ implies $v_1 p(R) v'_2$ due to rule (ORD). Transitivity then yields $(v_1, v_2) \in p(R)$.

Assume $v_2 \rightsquigarrow v'$ via a rule $l \mapsto r$, then $l = w'$, $r = w \sqcup w'$ where $(w, w') \in R$, according to definition of \mathcal{R} . Due to closure under linear combinations and

idempotency of \sqcup , we have $w \sqcup w' p(R) w' \sqcup w' = w'$, i.e. $r p(R) l$. Therefore, due to closure under scalar multiplication, $r \cdot (l \rightarrow v_2) p(R) l \cdot (l \rightarrow v_2)$, due to closure under linear combination $v_2 \sqcup r \cdot (l \rightarrow v_2) p(R) v_2 \sqcup l \cdot (l \rightarrow v_2)$. Keeping in mind the definition of \rightarrow , we can observe that $l \cdot (l \rightarrow v_2) \leq v_2$, and applying this we obtain $v_2 \sqcup l \cdot (l \rightarrow v_2) = v_2$, therefore $v' p(R) v_2$. Transitivity yields the claim for \sim^* .

- If $v p(R) v'$, then $\Downarrow v' \geq v$. This is the analogue to Theorem 3.8. We perform a proof by structural induction on the modified derivation rules of Table 1 (where (SYM) is removed and (REFL) is replaced by rule (ORD)).
- (REL) If $v p(R) v'$ because $v R v'$ then there exists a rewriting rule $(v' \mapsto v \sqcup v') \in \mathcal{R}$. Applying this rule shows that $\Downarrow v' \geq v' \sqcup v \geq v$.
- (ORD) If $v p(R) v'$ because of the ordering rule, i.e. $v \leq v'$, then $\Downarrow v' \geq v' = v$.
- (TRANS) If $v_1 p(R) v_3$ because of $v_1 p(R) v_2$ and $v_2 p(R) v_3$, then $\Downarrow v_3 \geq v_2$ implies $\Downarrow v_3 \geq \Downarrow v_2$. Furthermore $v_1 p(R) v_2$ inductively yields $\Downarrow v_2 \geq v_1$ and transitivity therefore yields $\Downarrow v_3 \geq v_1$.
- (SCA) If $v \cdot \ell p(R) v' \cdot \ell$ because $v p(R) p$, then $v \leq \Downarrow v'$ and Lemma 3.7 yields $\Downarrow (v' \cdot \ell) \geq (\Downarrow v') \cdot \ell \geq v \cdot \ell$.
- (PLUS) If $\overline{v} \sqcup v p(R) \overline{v'} \sqcup v'$ because $\overline{v} p(R) \overline{v'}$ and $v p(R) v'$, then $v \sqcup \overline{v} \leq (\Downarrow_{\mathcal{R}} v') \sqcup (\Downarrow_{\mathcal{R}} \overline{v'}) \leq \Downarrow (v' \sqcup \overline{v'})$, due to the monotonicity of \Downarrow .

□

Threshold Problem for Automata over the Tropical Semiring

Proof of Lemma 4.7

Proof.

- (i) Observe that \mathcal{A} is increasing and thus, if $\mathcal{A}(t_a(v))[x] = \infty$, $\mathcal{A}(t_a(\mathcal{A}(v)))[x] = \infty$ necessarily holds, too. Also note that $\mathcal{A}(t_a(v))[x] > T$ implies $\mathcal{A}(t_a(v))[x] = \infty$. Thus we can prove this lemma by showing the following: Let $I = \{x \in X \mid u[x] > T\}$, $x \in X$ and $a \in A$ such that $t_a(v)[x] \leq T$ be given, then $t_a(\mathcal{A}(v))[x] = t_a(v)[x]$.

All entries greater than T necessarily get mapped to ∞ .

We first compute:

$$t_a(\mathcal{A}(v))[x] = \bigsqcup \{\mathcal{A}(v)[y] \sqcap t_a[y, x] \mid y \in X\} = \min\{\mathcal{A}(v)[y] + t_a[y, x] \mid y \in X\}$$

and analogously

$$t_a(v)[x] = \min\{v[y] + t_a[y, x] \mid y \in X\}$$

Since we know that $t_a(v)[x] \leq T$, there must exist a $y \in X$ such that $v[y] + t_a[y, x] \leq T$. Observe that $v[x] > T$ and $t_a[y', x] \in \mathbb{N}_0$ for all $y' \in I$, therefore $y \notin I$. Thus

$$t_a(v)[x] = \min\{v[y] + t_a[y, x] \mid y \in X \setminus I\}$$

Now we can compute:

$$\begin{aligned}
t_a(\mathcal{A}(v))[x] &= \min\{\mathcal{A}(v)[y] + t_a[y, x] \mid y \in X\} \\
&= \min\{\min\{\mathcal{A}(v)[y] + t_a[y, x] \mid y \in X \setminus I\}, \min\{\mathcal{A}(v)[y'] + t_a[y', x] \mid y' \in I\}\} \\
&= \min\{t_a(v)[x], \min\{\infty + t_a[y', x] \mid y' \in I\}\} = \min\{t_a(v)[x], \infty\} = t_a(v)[x]
\end{aligned}$$

- (ii) First we show that if $o(v) \leq T$ it also holds that $o(\mathcal{A}(v)) = o(v)$. If $o(v) \leq T$ then $\min\{o[i] + v[i] \mid 1 \leq i \leq |X|\} \leq T$, so there is an index i where the minimum is reached and smaller than or equal T , i.e. $o[i] + v[i] \leq T$. Thus, since $+$ is increasing, $v[i] \leq T$ and therefore $\mathcal{A}(v[i]) = v[i]$. Since also \mathcal{A} is only increasing, we can conclude $o(\mathcal{A}(v)) = o(v)$.

It remains to be shown that $o(v) > T \Leftrightarrow o(\mathcal{A}(v)) > T$. Certainly, if $o(v) > T$ it must follow that $o(\mathcal{A}(v)) > T$, since $\mathcal{A}(v) \geq v$. So assume now, for the converse direction, that $o(\mathcal{A}(v)) > T$ holds. This means $\min\{o[i] + \mathcal{A}(v)[i] \mid 1 \leq i \leq |X|\} > T$ and assume $o(\mathcal{A}(v)) \neq o(v)$. Then there exists an index i where the minimum is reached, i.e. an index such that $o[i] + v[i] \leq o[j] + v[j]$ for all j . Since \mathcal{A} is only increasing the entries of a vector, it follows $o[i] + v[i] < o[j] + \mathcal{A}(v[j])$ for all j . Thus, $o[i] + v[i] < o[i] + \mathcal{A}(v[i])$, i.e. $v[i] < \mathcal{A}(v[i])$. It follows that $\mathcal{A}(v[i]) = \infty$. Then, $v[i] > T$ and since $o[i] \geq 0$ it follows that $o(v) > T$.

□

Lemma A.10. $p \bullet \sqsubseteq$ is b_2 -compatible, where $p: Rel_{\mathbb{S}^X} \rightarrow Rel_{\mathbb{S}^X}$ is the monotone function assigning to each relation R its pre-congruence closure.

Proof. By Lemma A.8 and Lemma A.9, \sqsubseteq and p are b_2 -compatible. It is easy to check that for all relations R, S , it holds that $b_2(R) \bullet b_2(S) \subseteq b_2(R \bullet S)$. Therefore, by Lemma A.6, $p \bullet \geq$ is b_2 -compatible. □

Proof of Theorem 4.8

Proof. Termination is obvious as there are only finitely many vectors with entries from the set $\{0, 1, 2, \dots, T, \infty\}$. We now prove soundness. Observe that the following is an invariant for the while loop at step (3).

$$R \subseteq b_2((p(R) \cup todo) \bullet \sqsubseteq)$$

since $\mathcal{A}(t_a(v'_1)) \sqsubseteq t_a(v'_1)$, i.e. $\mathcal{A}(t_a(v'_1)) \geq t_a(v'_1)$. If HKP_A returns *true* then *todo* is empty and thus $R \subseteq b_2(p(R) \bullet \sqsubseteq)$. By Lemma A.10, Theorem A.5 and Lemma A.7, $e_t \sqsubseteq v_1$, i.e. $T \geq \llbracket v_1 \rrbracket(w)$ for all $w \in A^*$.

The converse implication is more elaborated than its analogous in Theorem 4.3. Assume that HKP_A yields false, then a vector v'_1 was found such that $o(v'_1) > T$. This means there exists a word $w = a_1 a_2 \dots a_n$ such that

$$\begin{aligned}
v'_1 &= t_{a_n}(\mathcal{A}(t_{a_{n-1}}(\mathcal{A}(t_{a_{n-2}}(\dots \mathcal{A}(t_{a_1}(v_1)) \dots)))) \\
&\leq \mathcal{A}(t_{a_n}(\mathcal{A}(t_{a_{n-1}}(\mathcal{A}(t_{a_{n-2}}(\dots \mathcal{A}(t_{a_1}(v_1)) \dots))))))
\end{aligned}$$

Now we can apply Lemma 4.7 and eliminate all inner \mathcal{A} -applications, yielding $v'_1 \leq \mathcal{A}(t_{a_n}(t_{a_{n-1}}(t_{a_{n-2}}(\dots(t_{a_1}(v_1))\dots))))$. For easier reading we will now call $v' := t_{a_n}(t_{a_{n-1}}(t_{a_{n-2}}(\dots(t_{a_1}(v_1))\dots)))$. Since $o(v'_1) > T$, certainly $o(\mathcal{A}(v')) > T$ due to transitivity. Since \mathcal{A} is increasing, $o(\mathcal{A}(v')) \leq \mathcal{A}(o(\mathcal{A}(v')))$, which is, according to Lemma 4.7, $\mathcal{A}(o(v'))$. Due to transitivity of $>$ we can conclude $\mathcal{A}(o(v')) > T$, i.e. $\mathcal{A}(o(v')) = \infty$. According to the definition of \mathcal{A} , it follows that $o(v') > T$, therefore w is indeed a witness for the automaton not to respect the threshold. \square

Exploiting Similarity

Proof of Lemma 4.10

Proof. We will prove this inductively, by showing that $\llbracket v \rrbracket(w) \sqsubseteq \llbracket v' \rrbracket(w)$ for all $w \in \Sigma^*$.

- **Induction start** ($|w| = 0$): In this case, $\llbracket v \rrbracket(w) = \llbracket v \rrbracket(\varepsilon) = o(v) \sqsubseteq o(v') = \llbracket v' \rrbracket(\varepsilon) = \llbracket v' \rrbracket(w)$.
- **Induction hypothesis:** For all words w where $|w| \leq n$ it holds that $\llbracket v \rrbracket(w) \sqsubseteq \llbracket v' \rrbracket(w)$.
- **Induction step** ($n \rightarrow n + 1$): Let w be given such that $|w| = n + 1$. Then w can be written as $w = aw'$, $a \in \Sigma$, $w' \in \Sigma^*$. Since $(v, v') \in R$, there must exist (u, u') , $(v_1, v'_1), (v_2, v'_2), \dots, (v_m, v'_m) \in R$, $s_1, s_2, \dots, s_m \in \mathbb{L}$ such that $u = \sqcup \{v_i \cdot s_i \mid 1 \leq i \leq m\}$, $u' = \sqcup \{v'_i \cdot s_i \mid 1 \leq i \leq m\}$ and $t_a(v) \sqsubseteq u$, $u' \sqsubseteq t_a(v')$. Using monotonicity of \cdot and \sqcup (wrt. \sqsubseteq), we obtain the following two inequalities:

$$\begin{aligned} \llbracket v \rrbracket(w) &= \llbracket v \rrbracket(aw') = \llbracket t_a(v) \rrbracket(w') \sqsubseteq \llbracket u \rrbracket(w') \\ \llbracket u' \rrbracket(w') &\sqsubseteq \llbracket t_a(v') \rrbracket(w') = \llbracket v' \rrbracket(aw') = \llbracket v' \rrbracket(w) \end{aligned}$$

Now we can apply the induction hypothesis, keeping in mind that $|w'| = n$: Since $\llbracket v_i \rrbracket(w') \sqsubseteq \llbracket v'_i \rrbracket(w')$ for all $1 \leq i \leq m$. Applying again monotonicity of \cdot and \sqcup , as well as the definition of u and u' , we get $\llbracket u \rrbracket(w') \sqsubseteq \llbracket u' \rrbracket(w')$. Finally, transitivity yields $\llbracket v \rrbracket(w) \sqsubseteq \llbracket v' \rrbracket(w)$. \square

Proof of Lemma 4.11

Proof. First observe that since simulations are closed under union, there exists a greatest simulation relation on unit vectors.

- Due to the nature of the algorithm, it necessarily terminates after finitely many steps: In the beginning, R contains only finitely many pairs of vectors and in each iteration, either some pairs are removed from R , or R does not change, but in the latter case the algorithm terminates.
- The result is always a simulation relation, this can be seen as follows: Let R be the result of a run of the algorithm in Fig. 2 and $(v, v') \in R$. Then

- It holds that $o(v) \sqsubseteq o(v')$, because otherwise, the pair (v, v') would have been removed from R in the first foreach loop.
- Furthermore

$$t_a(v) \sqsubseteq \bigsqcup \{v_1 \cdot (v_2 \rightarrow t_a(v')) \mid (v_1, v_2) \in R\} =: u$$

Moreover, for all $(v_1, v_2) \in R$, $v_2 \cdot (v_2 \rightarrow t_a(v')) \sqsubseteq t_a(v')$ holds per definition of residuation. Therefore,

$$u' := \bigsqcup \{v_2 \cdot (v_2 \rightarrow t_a(v')) \mid (v_1, v_2) \in R\} \sqsubseteq t_a(v')$$

holds, as well. This means that (u, u') is a linear combination of R -vectors and $t_a(v) \sqsubseteq u, u' \sqsubseteq t_a(v')$.

- Now we will show inductively, that there cannot exist any greater simulation relation. The algorithm starts with the full cross-product of unit vectors and removes all pairs (v, v') where $o(v) \sqsubseteq o(v')$ does not hold – meaning that these pairs cannot be in a simulation relation at all. Thus, before we enter the nested foreach loops, R is a superset of (or equal to) the greatest simulation relation on α . We will now show that, whenever a pair of vectors is removed in the nested foreach loops, it cannot be contained in a simulation relation, therefore proving that after each execution of the nested foreach loops, R retains the property to be a superset of (or equal to) the greatest simulation relation on α .

Assume that (v, v') is an element of the greatest simulation relation $R' = \{(v_1, v'_1), (v_2, v'_2), \dots, (v_n, v'_n)\}$. Then there must exist multiplicands s_1, s_2, \dots, s_n such that $t_a(v) \sqsubseteq \bigsqcup \{v_i \cdot s_i \mid 1 \leq i \leq n\}$ and $\bigsqcup \{v'_i \cdot s_i \mid 1 \leq i \leq n\} \sqsubseteq t_a(v')$. From this it follows that for any given $1 \leq i \leq n$, it holds that $v'_i \cdot s_i \sqsubseteq t_a(v')$ and therefore $s_i \sqsubseteq \bigsqcup \{\ell \mid v'_i \cdot \ell \sqsubseteq t_a(v')\} = v'_i \rightarrow t_a(v')$, since it is included in the set. We can therefore compute, using $R' \sqsubseteq R$, which is the induction hypothesis:

$$\begin{aligned} u &= \bigsqcup \{v_1 \cdot (v_2 \rightarrow t_a(v')) \mid (v_1, v_2) \in R\} \\ &\geq \bigsqcup \{v_1 \cdot (v_2 \rightarrow t_a(v')) \mid (v_1, v_2) \in R'\} \\ &= \bigsqcup \{v_i \cdot (v'_i \rightarrow t_a(v')) \mid 1 \leq i \leq n\} \\ &\geq \bigsqcup \{v_i \cdot s_i \mid 1 \leq i \leq n\} \geq t_a(v) \end{aligned}$$

Thus, the if-condition $t_a(v) \not\sqsubseteq u$ in the second-to-last line does not evaluate to true, meaning that (v, v') will not be removed from R in the current iteration.

□

Proof of Lemma 4.12

Proof. Assuming all semiring operations (supremum, multiplication, residuation) consume constant time, the runtime of the algorithm can be analysed as follows: the for-loop in line (3) is executed $|X|^2$ many times, once for each element of R , which is

initialised as all pairs of unit vectors of dimension $|X|$, of which there are exactly $|X|$ many. The while-loop in line (4) is executed until R remains constant for one iteration. Since R contains at most $|X|^2$ many elements in the beginning (if no pair was thrown out in the preceding for-loop), and in each step of the inner for all-loop, (4.2.1), R remains either constant or a pair of vectors gets taken out of R . The for all-loop in line 4.2.1 is executed $|A|$ -times each time, and the inner for all loop is executed $|R|$ -times. While line (4.2.1.2) only takes constant time, line (4.2.1.1) takes $|X| \cdot |R|$ many steps. At worst, the time taken by the whole while-loop in line (4) therefore is $|A| \cdot \sum_{i=1}^{|X|^2} ((|X|^2 - i)^2 \cdot |X|) \in \mathcal{O}(|A| \cdot |X|^7)$, if in each iteration exactly one pair of vectors gets removed from R . Obviously, the latter loop dominates the former, so the run time is in $\mathcal{O}(|A| \cdot |X|^7)$. \square

Proof of Lemma 4.13

Proof. For soundness, we use the invariant $R \subseteq b_2(p'(R) \cup \text{todo})$, which allows to conclude that $R \subseteq b_2(p'(R))$. Now p' is not guaranteed to be compatible, but p'' defined for all relations R as $p''(R) = p(R \cup \sqsubseteq)$ is compatible by Lemma A.8 and Lemma A.6. By Lemma 4.10, $\preceq \subseteq \sqsubseteq$. By monotonicity of p , we have that $p'(R) = p(R \cup \preceq) \subseteq p(R \cup \sqsubseteq) = p''(R)$. By monotonicity of b_2 , $R \subseteq b_2(p''(R))$. By Theorem A.5 and Lemma A.7, $v_1 \sqsubseteq v_2$. For completeness, we proceed in the same way as in Theorem 4.3. \square

Proof of Lemma 4.14

Proof. For termination and completeness we reuse the same argument as in Theorem 4.8. For soundness we need to combine the proof of Lemma 4.13 and of Theorem 4.8: we use the invariant $R \subseteq b_2((p'(R) \cup \text{todo}) \bullet \sqsubseteq)$, which allows to conclude that $R \subseteq b_2(p'(R) \bullet \sqsubseteq)$. Now p' is not guaranteed to be b_2 -compatible, but p'' , as defined in the proof of Lemma 4.13 is. By Lemma A.6, $p'' \bullet \sqsubseteq$ is compatible, since \sqsubseteq is compatible. By monotonicity of p , we have that $p'(R) \bullet \sqsubseteq = p(R \cup \preceq) \bullet \sqsubseteq \subseteq p(R \cup \sqsubseteq) \bullet \sqsubseteq = p''(R) \bullet \sqsubseteq$. By monotonicity of b_2 , $R \subseteq b_2(p''(R) \bullet \sqsubseteq)$. By Theorem A.5 and Lemma A.7, $v_1 \sqsubseteq v_2$. \square

B Shortest Path Problem in Directed Weighted Graphs

The rewriting algorithm to compute the congruence closure over the tropical semiring is closely related to the Dijkstra algorithm to find the shortest paths in directed weighted graphs.

A *weighted directed graph* is a couple $G = (V, \text{weight})$ where $V = \{1, 2, \dots, n\}$ is the set of vertices and $\text{weight} : E \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ is a function assigning to each pair of vertexes v, w the weight to move from v to w . Intuitively, the weight is ∞ if there is no way (no edges) to go from v to w .

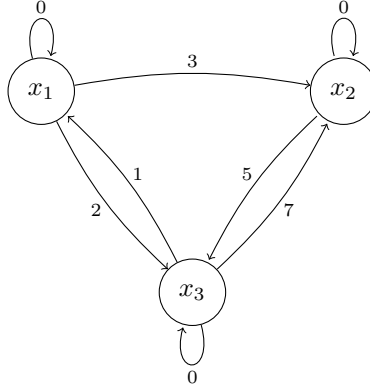
Definition B.1 (Rewriting System of a Graph). *Let $G = (V, E, \text{weight})$ be a weighted, directed graph and e_i be the i -th unit vector of dimension $|V|$ in \mathbb{T} and fix v_i to be the*

vector representing the outgoing arrows from the i -th vertex, i.e. $v_i[j] = \text{weight}((i, j))$. The rewriting system associated to G is $\mathcal{R}_G = \{e_i \mapsto v_i \mid 1 \leq i \leq |V|\}$.

Proposition B.2. For a graph G and vertex i , let $\Downarrow e_i$ be the normal form reached with \mathcal{R}_G . Then for all vertexes j , $\Downarrow e_i[j]$ is weight of the shortest path from i to j .

In order for rewriting to behave exactly like Dijkstra's algorithm, we need to always choose the rule with the smallest multiplicand that is applicable – i.e. the greatest one according to the order of the lattice. Choosing a rule corresponds to choosing a vertex to explore, determining the multiplicand corresponds to finding the weight accumulated from the starting vertex to the vertex to be explored next and applying the rule corresponds to updating the distances of all adjacent vertices that can be reached via a shorter path than the currently known shortest path. The following example shows the rewriting at work.

Example B.3. Consider the following directed graph:



This graph corresponds to the following rule system:

$$\mathcal{R} = \left\{ \begin{pmatrix} 0 \\ \infty \\ \infty \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ 3 \\ 2 \end{pmatrix}, \begin{pmatrix} \infty \\ 0 \\ \infty \end{pmatrix} \mapsto \begin{pmatrix} \infty \\ 0 \\ 5 \end{pmatrix}, \begin{pmatrix} \infty \\ \infty \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} 1 \\ 7 \\ 0 \end{pmatrix} \right\}$$

Now we can, for instance, determine the weights of the shortest paths to all vertices starting in the vertex x_3 as follows:

$$\begin{pmatrix} \infty \\ \infty \\ 0 \end{pmatrix} \rightsquigarrow 1 + \begin{pmatrix} 0 \\ \infty \\ \infty \end{pmatrix} \min \begin{pmatrix} 1 \\ 7 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \\ 0 \end{pmatrix} \rightsquigarrow 1 + \begin{pmatrix} 0 \\ 3 \\ 2 \end{pmatrix} \min \begin{pmatrix} 1 \\ 7 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 0 \end{pmatrix}$$

Afterwards, no more rewriting rule is applicable.

Note that rewriting is non-deterministic and we could have chosen another path. If we choose to apply the rule that has smallest coefficient, we effectively simulate Dijkstra's algorithm.